University of Alberta

MOBILE ROBOT OBSTACLE AVOIDANCE USING LEARNING CLASSIFIER SYSTEMS

by

Sa Li

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**.

Department of Electrical and Computer Engineering

Edmonton, Alberta

Fall 2007

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-33014-2
Our file Notre référence
ISBN: 978-0-494-33014-2

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

During the past few decades there has been much development of model-based approaches to autonomous mobile robot obstacle avoidance. Much of the work centre on navigation using artificial intelligence (AI) techniques. The goal of this thesis is to explore alternative approaches in the evolution of robot intelligence, rather than its prior design. It is proposed to use a method of evolutionary computing (EC)–the learning classifier system (LCS)–that possesses salient properties necessary for learning. Several LCS-type techniques are used to evolve obstacle-avoidance rules for mobile robot navigation. The robot learns these rules via feedback from the environment, available as sonar readings. Conventional LCSs (Michigan-style LCSs) show evidence of shortcomings: they become trapped in local minima, they lose desirable rules, and they favor rules that are too general. Enhancements proposed in this thesis overcome these limitations, and can thus handle more complex situations than existing devices. The inclusion of a dynamics feature helps to prevent oscillations of the learning process and makes robot learning converge quickly after each collision. To combine the structural learning ability of an LCS and the continuous adaptive characteristics of a neural network (NN), an LCS-like NN is constructed by mapping an LCS onto an NN. The system employs a genetic algorithm (GA) in a distinct way which demonstrates its co-adaptive characteristics. The LCS-like NN described in this thesis proves to be effective by evolving a diverse population of hidden nodes that compete to perform robot actions.

Acknowledgements

I dedicate this thesis to my wife Li Du, and my daughter Nicole X. Li, my mother Luan Wang, and my father Fengshan Li. I thank them for their support, understanding and encouragement.

I express appreciation and gratitude to my supervisor Dr. Petr Musílek, for invaluable guidance and contributions to the work. Dr. Musílek was patient and understanding throughout the completion of this thesis. He gave support and encouragement through difficult periods of the research, bringing a spirit of exploring science to the project. The completion of this thesis would be impossible without his close supervision and devotion to education.

The scholarship granted by University of Alberta is greatly appreciated. Also, I thank members of the Facility for Advanced Computational Intelligence and Applications (FACIA) for help and encouragement.

Table of Contents

1 Introduction 1

1.1 Motivation and Background 1

1.1.1 Motivation 1

1.1.2 Background 2

1.2 Thesis Overview and Contributions 3

1.2.1 Thesis Overview 3

1.2.2 Major Contributions 4

2 Literature Review 6

2.1 Computational Intelligence in Robot Navigation 7

2.1.1 Neural Networks 7

2.1.2 Fuzzy Systems 8

2.1.3 Evolutionary Computing 9

2.2 Learning Classifier Systems in Robot Navigation 13

2.2.1 An Overview of Learning Classifier Systems 13

2.2.2 Applications of LCS in Robotics 15

3 Mobile Robotics 19

3.1 Robot Navigation 20

3.2 Robot Sensing 23

3.3 Robot Controllers Based on Computational Intelligence 25

3.3.1 Neural Controllers 25

3.3.2 Fuzzy Controllers 28
3.3.3 Evolutionary Robot Controllers 31
3.4 Issues and challenges 34

4 Learning Classifier Systems **36**
4.1 Machine Learning 36
4.1.1 Definitions 36
4.1.2 Objectives 37
4.1.3 Issues 38
4.1.4 Approaches 39
4.2 Genetic Algorithms 41
4.2.1 History 41
4.2.2 Fundamentals and Working Principles 42
4.2.3 Parameters of Genetic Algorithms 48
4.3 Operation of Learning Classifier Systems 48
4.3.1 Rule and Message Subsystem 49
4.3.2 Apportionment of Credit Subsystem 52
4.3.3 Rule Discovery Mechanism 54
4.4 Extended Classifier System (XCS) 55
4.4.1 The Performance Subsystem 57
4.4.2 The Reinforcement Subsystem 57
4.4.3 The Rule Discovery Subsystem 58

5 LCS-based Robot Controller **60**
5.1 Representation: Encoding and Decoding 64
5.2 Matching Process 66
5.3 Credit Assignment Process 67
5.4 Rule Discovery Process 68
5.5 Implementation 69
5.6 Simulation Results 71

6 Enhanced LCS Controller with Perceptual Memory **75**

6.1 Shortcomings of the LCS Controller 75

6.1.1 Local Minima 75

6.1.2 Rule Destruction 76

6.1.3 Generalization 76

6.2 Enhanced LCSs 77

6.2.1 Enhancements 77

6.2.2 Simulations 78

6.2.3 Analysis 85

6.3 An LCS with Perceptual Dynamics 86

7 LCS-like NN Robot Controller **92**

7.1 An LCS-like NN 92

7.1.1 Mapping LCS into NN 93

7.1.2 Execution of an LCS-Like NN 95

7.2 An LCS-Like NN Controller 97

7.3 Parameters of the GA 103

7.3.1 The Number of Hidden Layer Nodes 103

7.3.2 Probability of Crossover 105

7.3.3 Probability of Mutation 107

8 Conclusions and Future Work **109**

8.1 Conclusions 109

8.2 Future Work 111

Bibliography **113**

List of Figures

3.1 The basic process of the robot navigation 22
3.2 The MLP robot controller architecture 25
3.3 The configuration of sonars 28
3.4 Input membership functions for each input 29

4.1 The process of reinforcement learning 40
4.2 Flowchart for a GA 43
4.3 Examples of chromosomes 44
4.4 Single-point crossover operator 47
4.5 Bit mutation operator 47
4.6 Block diagram and data flow of a typical LCS 50
4.7 A schematic illustration of XCS reproduced from [113] 56

5.1 Pioneer 2-DX robot 61
5.2 The layout of a SRIsimulator 62
5.3 The layout of a pilaster environment 63
5.4 The layout of a polygon environment 63
5.5 The placements of sonar 64
5.6 Implementation cycle of obstacle avoidance by a LCS controller 70
5.7 The trajectory of robot at the beginning of learning (pilaster) 71
5.8 The trajectory of robot at the beginning of learning (polygon) 71
5.9 The trajectory of the robot in an equilibrium state (pilaster) 72
5.10 The trajectory of the robot in another equilibrium state (pilaster) 72

5.11 The trajectory of the robot in an equilibrium state (polygon) 73
5.12 The trajectory of the robot in another equilibrium state (polygon) 73
5.13 Non-collision-cumulation with large horizontal scale (pilaster) 74

6.1 Implementation cycle of obstacle avoidance by an enhanced LCS controller 79
6.2 Robot trajectory using enhanced LCS during the learning period (pilaster) . 82
6.3 Robot trajectory using enhanced LCS during the learning period (polygon) . 82
6.4 Robot trajectory using enhanced LCS after the learning period (pilaster) . . 83
6.5 Robot trajectory using enhanced LCS after the learning period (polygon) . . 83
6.6 Non-collision-cumulation (pilaster) 84
6.7 Non-collision-cumulation (polygon) 84
6.8 Moving average of collision occurrence frequency by enhanced LCS (pilaster) 88
6.9 Moving average of collision occurrence frequency by enhanced DCS (pilaster) 88
6.10 The probability distribution of CCSs by enhanced LCS (pilaster). 89
6.11 The probability distribution of CCSs by enhanced DCS (pilaster) 89

7.1 A LCS as a Competitive NN 93
7.2 Implementation cycle of obstacle avoidance by an LCS-like NN controller . 100
7.3 Robot trajectory using LCS-like NN after the learning period (pilaster) . . . 102
7.4 Robot trajectory using LCS-like NN after the learning period (polygon) . . 102
7.5 Non-collision-cumulation (pilaster) 103
7.6 Non-collision-cumulation with population size 100 (polygon). 105
7.7 Non-collision-cumulation with population size 200 (polygon). 105
7.8 Non-collision-cumulation with population size 300 (polygon). 105
7.9 Non-collision-cumulation with population size 500 (polygon). 105
7.10 The effect of different P_c on learning time 107
7.11 The effect of different $P_{m,spec}$ on learning time 107

List of Acronyms

Acronyms	Definition
ER	Evolutionary Robotics
EC	Evolutionary Computing
LCS	Learning Classifier System
NN	Neural Network
DNN	Dynamic Neural Network
ALVINN	Autonomous Land Vehicle In a Neural Network
RNN	Recurrent Neural Network
GA	Genetic Algorithms
SAGA	Species Adaptation Genetic Algorithm
MLP	Multi-layer Perceptron
BP	Back-propagation
FL	Fuzzy Logic
FC	Fuzzy Control
TOF	Time-Of-Flight
CCD	Charge-Coupled Device
GP	Genetic Programming (GP)
ES	Evolutionary Strategies
EP	Evolutionary Programming
FSM	Finite-State Machine
ML	Machine Learning

RL	Reinforcement Learning
MSE	Mean Square Error
S-R	Stimulus-Response

List of Symbols

Symbol	Definition
Q	the states of environment
P_f	payoff
W_{in}	weights between hidden and input layers
W_{out}	weights between hidden and output layers
U_{hid}	input of hidden layer
Y_{hid}	hidden neuron activation function
U_{out}	input of output layer
Y_{out}	output neuron activation function
α	slope parameter
μ, β	learning rate
P_i	probability of selection
P_c	probability of crossover
P_m	probability of mutation
P_{cum}	cumulative probability
λ	conditional part of a LCS
φ	action part of a LCS
F_m	match function
C_i	classifier i
$S_i(t)$	classifier strength
$B_i(t)$	bid of a classifier

ρ	risk factor
q_i^{λ}	specificity
$R_i(t)$	reward
$\mathrm{Tax}_{\mathrm{life}}$	tax of life
$\mathrm{Tax}_{\mathrm{bid}}$	tax of bid
γ	discount factor
ε	prediction error
k_j	accuracy
$\mathrm{X_i}$	sonar reading

Chapter 1

Introduction

1.1 Motivation and Background

1.1.1 Motivation

The development of techniques for autonomous mobile robot navigation in a real world environment is a major trend in current robotics research. This trend is motivated by the potential applications of mobile robots in a variety of domains. There is a gap between the available technology and the growing demand for new applications. Lacking flexibility and autonomy, current industrial robots typically perform pre-programmed operations in highly structured environments. They are unable to operate in new environments or handle accidental situations. Consequently there is an emerging market for truly autonomous robots. According to market requirements, possible applications can be divided into four main categories:

1. Intelligent robots: to provide service for offices, hospitals, and factory floors (e.g., mail delivery in an office building, food delivery in a hospital, or security surveillance of a warehouse environment);
2. Maintenance robots: to operate in hazardous or inaccessible areas (e.g., fighting fires on ships or surveying ocean depths);
3. Domestic robots: for cleaning or entertainment (e.g., Sony's Aibo);

4. Assistance robots: to act as aides for handicapped individuals (e.g., semi-autonomous vehicles assisting disabled people).

Early mobile robots might have been found on outer space or exploring underwater environments. In the future, mobile robots will have the potential to influence, directly or indirectly, nearly every aspect of our daily lives.

1.1.2 Background

In each of the above applications, a robot is required to accomplish specific tasks. A mobile robot must be able to navigate its environment in order to carry out assignments, therefore mobile robots need the capability to explore and learn about their environments. The work described in this thesis is concerned with the problem of an autonomous mobile robot wandering in a strange environment while avoiding obstacles.

The basic problem of mobile robot navigation is to navigate from a start location to an end location without any collisions. Existing mobile robot controllers can be grouped as follows [116]:

- Model-based controllers (the robot controller uses a model of the environment to navigate);
- Senors-based controllers (the robot controller uses only sensory information);
- Hybrid controllers (the robot controller uses both environmental and sensory data).

This thesis intends to explore the sophistication of evolved machine intelligence. The focus of the work is in the field of evolutionary robotics (ER). Evolutionary robotics [47] applies evolutionary computing (EC) methods to evolve a population of controllers for use in robots. One important problem in mobile robot navigation is the need to cope with the large amount of uncertainty when dealing with natural environments. In the EC domain, the LCS is a machine learning (ML) system that evolves rules in order to improve its performance in a changing environment [21] – these features can address the problem of environmental uncertainty. It is an online learning strategy where the robot controller is applied to navigational behaviours, such as obstacle avoidance or wall following. Dur-

ing the learning, sonar information is collected at each moment, a classifier is associated with an action to be taken in each specific state, and a RL algorithm is applied to execute rewards/punishments via feedback from the environment. Over a number of iterations, an optimal policy is learned. Since existing mobile robots are not completely autonomous, one of the main goals of this thesis is to use LCS to create a truly autonomous robot without any hand-design factors.

Another approach for dealing with uncertain and dynamic environments is based on NN. A NN is the multi-layer perceptron (MLP) trained with a back-propagation (BP) learning algorithm [34]. The NN is embedded in a traditional control scheme to navigate a robot from a start position to a goal position. The principal function of such a control scheme is to learn the parameters (e.g., connection weights) that lead to stable control. However, a NN is employed as little more than a function approximator with generalized properties in this kind of application because it relies on a priori knowledge of obstacle localization for path planning among obstacles inside the arena of robot motion. Therefore a detailed model of the environment needs to be built. In most cases, this model is impractical since it is difficult to plan a precise path among unknown obstacles in real and unstructured environments. An alternative approach is to design a NN with specialized learning rules. From this perspective, a NN can be combined with an LCS to control autonomous mobile robots. Later in this thesis, an LCS-like NN is used to construct a robot controller. By mapping an LCS onto an NN, the flexibility of the LCS is improved by the universal approximation ability of the NN without impairing its learning ability.

1.2 Thesis Overview and Contributions

1.2.1 Thesis Overview

The thesis is organized as follows.

Chapter 2 provides a brief review of the origins of behavioural robot control. Research in methods of NN control, fuzzy control (FC), and ER are summarized, and LCS applica-

tions to robot navigation are investigated in a literature review.

Chapter 3 presents the problem of mobile robot navigation; it describes the development of robot sensory systems and the challenges of this work.

In Chapter 4, an introduction to ML is provided and the genetic algorithm (GA), a key component of an LCS, is described. The focus is on what an LCS is and how it learns. In addition, the extended classifier system (XCS), a system that uses prediction accuracy as a basis for calculating classifier fitness, is briefly introduced for later comparison.

In Chapter 5, the application of LCS to robot navigation is described, along with simulation results showing some shortcomings of the conventional LCS.

In Chapter 6, proposals to overcome the limitations of LCS include several extensions for existing systems and a feature known as perceptual dynamics. The benefits of these modifications are demonstrated using additional experiments, and a comparison of an extended system with an XCS is made.

Chapter 7 describes the application of an LCS-like NN to robot navigation and reports simulation results. The parameters of GAs in LCS-like NNs are investigated.

Chapter 8 concludes the thesis and provides an outline for future work.

1.2.2 Major Contributions

The major contributions of this thesis can be summarized as follows:

1. Several enhancements to LCS are proposed in Chapter 6: to allow operation in a more complex and non-symmetric environment, extensions to a Michigan-style LCS [43] are made that effectively enhance its performance [59, 74].
2. A dynamical feature is included in LCS: addition of perceptual dynamics to LCS helps to prevent oscillations of the learning process and allows it to converge quickly after each collision [59]. This part is described in Chapter 6.
3. A novel robot encoding and decoding scheme is proposed in Chapter 5: sonar signals are continuously transformed into binary/real values by a thresholding/scaling process. To translate the action of a robot controller into usable motor commands,

a decoding mechanism is introduced. The turning angles are encoded symmetrically so that the encoding scheme has more entries for actions with smaller turns, increasing the probability that the robot will move ahead [59, 60, 74].

4. A NN model based on the structure of robot inputs/outputs is developed in Chapter 7: this NN model fully reflects the working principle of an LCS, and thus combines the learning ability of LCS with the universal approximation properties of a NN [60].

Chapter 2

Literature Review

In the late 1940's, Grey Walter carried out pioneering research on autonomous mobile robotics [110]. Walter constructed a three-wheeled, turtle like, autonomous robot that showed simple behaviours. This robot was a purely analog device with very simple reactions, however, it demonstrated the speculative tendency to explore its environment. Walter's work inspired subsequent generations of robot researchers by revealing that simple reflex actions and their consequent behaviours arise from neural interconnections. In contrast to traditional artificial intelligence approaches that focus on high level cognitive tasks, behaviour-based robotics is a paradigm that provides robots with the sensorimotor abilities observed in animals. The basic idea underlying behaviour-based robotics is to collect simple reactive behaviours [1] that operate in parallel and use them to produce more complex forms of activity.

In the 1980's Braitenberg [7] developed a behaviour-based robot controller with direct connections of varying complexity from sensors to effectors (drive motors). Braitenberg classified simple reactive mobile robot controllers based on the degree of complexity of the mapping of primary sensor inputs to final effector outputs. Modern behavioural robotics employs a wide variety of conceptual and structural robot control architectures [72]; the most prominent is the subsumption architecture proposed by Brooks [9] which builds up a robot control system layer by layer. Each layer in the Brooks model is responsible for a new

robot behaviour which receives inputs directly from sensors and sends outputs directly to motors. These behaviours can interact with each other, but higher-level layers can suppress or inhibit interactions between lower-level layers.

Behaviour-based controllers provide robots with the capacity to respond quickly to an unknown environment. This reactivity is a necessary component of any mobile robot. However, to complete more sophisticated tasks – for instance, to learn the spatial structure of the environment and use this knowledge for navigation – additional abilities are needed. Controllers that produce motor outputs as a direct result of sensor inputs, without an explicit internal environment model, are considered "model-free" [3]. Model-free controllers evolve through interactions with the environment. By contrast, human designers incorporate personal knowledge (internal complex world model) to construct "hand-designed" controllers. Model-free autonomous mobile controllers are able to execute rapid online modifications to robot behaviours. The main contributions in this area are discussed in the following sections.

2.1 Computational Intelligence in Robot Navigation

2.1.1 Neural Networks

NNs, which imitate real biological systems with adaptive abilities [92], have been considered for controlling mobile robots [73]. A NN is a computing device consisting of numerous simple elements wired together in parallel – loosely similar to the human brain. Unlike a conventional computer, which produces results based on algorithms (rules to be followed sequentially), a NN acts as an adaptive filter; it is trained by inputs and the desired responses to those inputs. The information changes the way the network is connected so that the next similar input can produce a similar correct output [34].

To accomplish the task of guiding robot navigation, a NN is designed to map sensory input to robot motor commands. A variety of NN structures and learning techniques have been proposed. Most of the work to date uses feedforward MLP networks which act as

a class of universal approximators [44]. The learning rules include BP [84–87, 95] and EC [55, 80], among other methods; the BP algorithm is probably the best-known and most often used. The BP algorithm is based on the steepest descent approach: the synaptic weights in the network are updated in proportion to the computed error at the output of the network; the computed error is the difference between actual and desired output [34]. An application of the NN controller is the ALVINN (Autonomous Land Vehicle In a Neural Network) system [84–86]. ALVINN is a BP network designed to map digital video images into steering commands. The training data (a human driver's reactions) is collected while the vehicle is driven by a human driver. The system has been successfully implemented on a real vehicle.

2.1.2 Fuzzy Systems

Environmental knowledge is generally incomplete, uncertain, and approximate. Perceptually acquired information is noisy and the dynamics of a real world environment cannot be accurately predicted; objects may move and the environment can be modified over time. In addition, a robot's hardware execution is not completely reliable; e.g., odometric errors accumulate due to the slip of robot wheels. A robot needs to make decisions and carry out actions in real time [119]. Classical planning approaches cannot adequately handle such situations. Alternative systems using principles of FC [8,24,77,90,103–105,117,119] have the advantage that they can process uncertain data.

Fuzzy logic (FL) was conceived by Lotfi Zadeh in 1965 as a mathematical way to represent uncertainty in the surrounding world [118]. It can provide the means to process uncertain data by allowing partial set membership rather than crisp set membership. In two-value logic, a state is either true or false, but not both. It is convenient to mathematically map the possible truth-values into a discrete domain, such as {0, 0.5, 1} for {false, maybe true, true}. In FL, a state may be true or false or have an intermediate truth-value, such as "maybe true". In other words, FL extends the idea of multi-valued logic allowing for continuous truth-values on the interval [0, 1].

FC, derived from fuzzy logic and fuzzy set theory, was not developed until the 1970's due to insufficient computing capability. Since it expresses and implements human knowledge in the form of linguistic *if-then* rules, FC can adequately cope with nonlinear systems that are difficult or impossible to model analytically [52]. Thus control systems previously considered infeasible for automation can now be built under FC. Every rule has two parts: an antecedent part, [*if* ...], and a consequent part, [*then* ...]. The antecedent part describes the system state that is used to activate the rule; the consequent part is the action the control system will take. The process states and control variables are called linguistic variables. A linguistic variable can take linguistic values of different magnitudes, graphically represented by continuous functions called membership functions.

There are four major parts in a fuzzy controller [24]:

1. The fuzzification interface defines the control objectives and criteria, determines input and output relationships, and chooses a minimum number of variables, mapping them into linguistic variables.
2. The knowledge base is a part of the systems that contains a series of *if* ... *then* ... rules that define the desired system output response for given input conditions. Membership functions and control rules are decided by experts, based on their knowledge of the system.
3. The decision-making logic (inference) treats a fuzzy set as a fuzzy proposition. One fuzzy proposition can imply another, and two or more fuzzy propositions can be combined by a connectivity relation to convert input information into output membership functions.
4. The defuzzification interface converts fuzzy outputs into crisp (non-fuzzy) values.

2.1.3 Evolutionary Computing

Charles Darwin described the process of natural selection in *The Origin of Species* [15]. Characteristics that enable an individual to survive can be passed to offspring during reproduction; individuals with poor characteristics die off. The species takes advantage of

this phenomena to become gradually stronger. EC is a research area within computer science and it attempts to simulate this natural process of "survival of the fittest"; that is, the "trial and error" style of problem solving is implemented in the evolutionary computing process [22].

EC techniques can be classified into three main areas [21]: *GA* [40], *evolutionary strategies* (ES) [89] and *evolutionary programming* (EP) [27]. These techniques were originally developed separately, but since the 1990s have been considered different representatives of one technology [22].

GAs involve manipulating fixed-length bit strings (genotypes) which represent candidate solutions to the problem being studied. A population of genotypes are evaluated at each generation; the probability of a genotype being picked as a parent is proportional to its fitness. Offspring are created through genetic operators: crossover and mutation. Since GA is of particular interest in this thesis, a more detailed description is given in Chapter 4.

ES was originally developed for solving problems in hydrodynamics, where n-dimensional real-valued vectors represent candidate solutions. In contrast with the typically fixed genetic operators used in GA, the mutation operators used in ES are under evolutionary control. Both the variances and co-variances of the mutation operator probability density distributions are specified genetically, and allowed to change during evolution.

EP attempts to simulate intelligent behaviour by means of finite-state machines (FSM). Typically, each member of the current population of size N produces one offspring through mutation, with the next generation of N genotypes being selected from the $2N$ current and offspring genotypes. Tournament selection is usually applied to select the genotypes propagated to the next generation.

All three types of EC systems share the following components [65]:

- A genetic representation for potential solutions to the problem (the string with fixed or variable length bits, expression tree, etc);
- A method to create an initial population of individuals ;
- A fitness function that plays the role of the environment, rating solutions in terms of their "adequacy";

- Operators that determine the composition of offsprings;
- Values for various parameters that the evolutionary algorithm uses (solution population size, probabilities of applying genetic operators, etc).

Recently, there has been great interest in ER – the use of artificial evolutionary algorithms for the design of autonomous mobile robot controllers. ER is a new technique for the automatic creation of autonomous robot controllers that do not require hand-designed or in-depth human knowledge of environments in which the robots move. "Inspired by the Darwinian principle of selective reproduction of the fittest, ER regards robots as autonomous artificial organisms that develop their own skills by interacting with the environment and without human intervention" [79]. That is, experts precisely specify the task that a robot is desired to complete, but leave the course by which the task is accomplished to an evolutionary process. Since human designers may not be able to design controllers beyond a certain complexity, specific technologies must be developed to enable controllers to produce more complex behaviours automatically. One way to accomplish this is to evolve the controller that displays the desired behaviours in the search space of all possible controllers.

In ER, an initial population of artificial chromosomes, representing the control scheme of a robot, is randomly initialized and placed in the environment. The robot is then free to act (move, look around, manipulate) in terms of its genetically specified controller while its performance is automatically evaluated. The fittest robots have the chance to reproduce by swapping parts of their genotypes. The evolution process is repeated until the "birth" of a robot that satisfies the predefined criteria [79].

The tremendous growth of computer speed since the early 1990s has promoted the development of ER. Throughout 1990's, a large amount of experimental research was published in this field; much of the work was duplicated. In 1992, Koza [55] applied GP to develop a robot controller for a wall-following task. In 1994, Nolfi *et al.* reported experiments in obstacle avoidance using a recurrent neural network (RNN) in a two-wheeled robot [80]. In this case it took 60 hours to evolve desired behaviours. Another area of ER research is the navigation of legged robots. Several groups evolved and tested controllers for walking and obstacle avoidance on legged robots [25, 33, 49, 58].

Harvey proposed a new approach in 1992 – Species Adaptation Genetic Algorithm (SAGA) which particularly handles the GA dynamics by allowing the genotype lengths to increase [35]. Harvey's group has contributed to a series of works in the ER area [13, 35–39, 46, 48, 50, 69, 107, 108]. In other studies: Cliff *et al.* described the evolution of NN controllers for a simulated two-wheeled robot that used simulated photo detectors and tactile sensors in 1993 [13]; Jakobi described the use of minimal simulation to evolve behaviours in an eight legged robot with sixteen actuator motors in 1998 [49]; and Harvey *et al.* provided an overview of his group's work [39].

In [96], a fully embedded distributed evolutionary system was proposed to achieve collision-free-navigation in a few hundred trials. This supports the theory that the embedded evolution concept can be applied to the morphology and control circuit of a population of robots in real time. In [62], a software framework to create the vision subsystem of a reactive obstacle avoidance algorithm for a mobile robot by using GP is presented. The training set of environmental images was collected from a robot camera and evolving programs were evaluated on this fixed training set and compared to hand-designed solutions. The best evolved programs were then used to create obstacle avoidance programs which were combined with other systems to control a robot.

The roots of ER are:

- behavioural robotics (as mentioned at the beginning of this chapter);
- EC;
- reinforcement learning (RL).

In many cases, RL using high-level task performance feedback is applied to the evolution of robot controllers. Such feedback is required for the evolution of complex and non-trivial behaviours since a direct error-feedback signal is usually unavailable. In [111], idiotype networks are integrated with an RL algorithm for constructing a robot navigation controller. Experiments demonstrated that the proposed controller could deal with highly constrained navigation problems where robots have very little space to move. Similarly, in [2, 51], approaches that combined the principles of NN, RL, and evolutionary methods are developed to evolve topologies of the NN; the controllers then have the capacity to

explore the environment, avoiding risky configurations.

EC and RL are powerful approaches capable of generating autonomous learning in robot navigation problems. A well-established algorithm that combines these techniques is the LCS. LCS is a dynamic, rule-based system that utilizes RL to learn by example and induction. LCS is flexible and structurally adaptive compared with other computerized intelligent systems.

2.2 Learning Classifier Systems in Robot Navigation

2.2.1 An Overview of Learning Classifier Systems

The LCS, invented by John Holland [43], is a parallel rule production system which uses RL to change the fitness of the rules contained in the system and a GA to search the rule base for new, useful, and general rules. LCS is primarily used in situations where a system is evolved in responding to the current state of the environment, suggesting a response that maximizes a reward from the environment.

There are two type of LCSs. In one type, each individual in the population represents a single rule. This approach was developed by Holland and Reitman at the University of Michigan, so it is called the "Michigan" approach. This method works well in situations where the LCS operates entirely in a stimulus-response (S-R) mode; that is, there are no internal messages that affect which classifier fires. In an LCS with internal messages, the next rule expressed is linked to previous rules through internal messages; the linked rules form rule chains leading to a complex sequence of actions. In the second type of LCS, each individual represents an entire set of rules. This approach was developed at the University of Pittsburgh and is thus called the "Pitt" approach. Since each GA population member must be completely evaluated as a rule set, the second type of LCS requires a large computational cost and this may preclude online learning in many situations.

The Zeroth-Level classifier system (ZCS) was proposed by Wilson [112] to increase understandability and performance while keeping much of the original framework of the

Michigan-style LCS. It is a Q-learning-like system [68] which provides a viable foundation for building toward the aims of the Michigan-style LCS. The major difference is that the ZCS does not have an internal message list, and therefore there is no explicit arrangement for transmitting information before internal execution cycles. The GA preserves rules according to strength and rewards; the ZCS preserves only rules in niches that lead to maximum reward [22]. This feature is acceptable for many applications, but as Wilson indicated [112], the use of stochastic selection in the presence of highly fit initial solutions may lead ZCS to prematurely converge local minima before the entire space can be properly explored and stable rule bases formed with respect to each niche.

Closely following ZCS, the XCS, a classifier system based on accuracy, was introduced by Wilson [113]. XCS has many features in common with ZCS. The primary distinguishing feature of XCS is that the fitness of classifier is based on the accuracy of classifier payoff prediction instead of classifier strength [54]. Other differences between the Michigan-style LCS and XCS include the use of a niche GA, a Q-learning-like strength update mechanism, and a deletion mechanism which intends to balance classifier allocation among niches. Details of the XCS are introduced in Chapter 4.

In the different versions of classifier systems, inputs and outputs are generally represented by binary values. However, this kind of representation may not be appropriate in many situations. In some cases, the state of the environment and actions of the system are better described by real values. The fuzzy classifier system (FCS) is a natural real-valued variables extension to represent rules using fuzzy logic instead of crisp matching. FCS brings several powerful modern computational paradigms together; this architecture allows for a rule-based system to interact with a continuous real-valued environment via FL [82, 83]. The primary advantage of using fuzzy sets for the membership functions is that they can handle the noise better than crisp sets. This feature is especially useful when the system's perceptual information is inaccurate, as in robotics applications where sensor values are unreliable [22].

LCS is widely viewed for adaptive systems capable of online learning in rapidly changing environments. In recent years, several applications of LCS have been reported. Most

reports fall into three major areas [56]: autonomous robotics, data mining, and computational economics. Several researchers have also applied LCS to solve specific problems. In [12], an LCS for automatic global operation of canal gates was developed and tested through simulation modelling. Each rule in the population is trained to learn appropriate operational responses to unsteady hydraulic conditions in a canal system. [29] examines the effects of missing features in input messages on the performance of an LCS, and applications in the area of medical data analysis are discussed.

2.2.2 Applications of LCS in Robotics

Autonomous robotics is an important test-bed for LCS models. Several reasons for employing a Michigan-style LCS to create an autonomous robot rather than using a purely evolutionary approach, NNs, or a pure RL approach are:

- The addition of RL provides the robot with the online learning ability that a purely evolutionary approach lacks.
- Generalization is achieved with LCS through the evolution of general rules that are quite useful in many cases. For instance, one general rule can match many perceptual inputs.
- LCS is rule-based and contains a GA. It is unnecessary to render the continuous space into a discrete Markov space as when an RL approach is implemented for robotics. This process of RL involves either a priori decisions about the problem which may be wrong, or the addition of function approximation mechanisms such as NNs [102]. This can be avoided by employing an LCS.

In [18] Dorigo and Colombetti conducted extensive work in the area of LCS-based robot learning. They addressed some of the problems of the Michigan-style LCS and presented a parallel LCS, ALECSYS, to several robotics applications reporting interesting results. Further, they demonstrated a behavioural engineering approach which allows for hierarchical decomposition of the learner to aid learning. In their experiments with an "AutonoMouse" robot, environmental messages are collected in binary format; each bit in the

condition part of the rule is set to tell if there is a light located in front of one of the robot sensors. Four types of movement are used to control the motors: 1 step forward, stop, 1 step backward, 2 steps forward. Although the hierarchical decomposition work in [18] is impressive, it relies on a priori knowledge of the problem that instructs the hierarchical decomposition into behavioural modules, and the training policy used to reward the intention.

In [101], Stolzmann presented results using a Khepera robot to explore latent learning with the Anticipatory Classifier System (ACS). ACS is an enhancement of XCS with a degree of environmental model learning by the addition of anticipations. The robot and its environment are simulated with the open mobile robot simulator *Webots*. Lukins [61] developed a simple, complete robotic architecture called *FACADE*, consisting of a number of agents that are implemented as Stolzmann's ACS trained in basic behavioural responses. *FACADE* has not enriched ACS in the aspects of algorithm, but presents an experimental framework that can create a dynamic response in an animate. In order to realize another function of the *FACADE*–the ability to arbitrate between behavioural agents–a collection of behaviours need to be trained in a simulated environment. However, the training leads to an environment-dependent performance.

Hurst *et al.* [45] implemented Wilson's ZCS system for an obstacle avoidance task. The controller model is designed as "event" based: "*Instead of the robot being assigned fixed length actions, continuous actions are taken. These actions are taken until an "event" occurs. An event can be thought of as a change of state ...*" [45]. The authors then introduce a novel LCS–a temporal classifier system (TCS), a form of ZCS that contains a different idea of action persistence, and the RL algorithm factors in time. A simple light-seeking task on the robot platform "Linuxbot" using TCS is presented, and the simulation results show desirable adaptive characteristics for the use of LCS on real robots.

Pipe *et al.* [82,83] reported on experiments designed for an autonomous rule acquisition algorithm for fuzzy controllers. The algorithm is a FCS which operates on the rules of fuzzy controllers with predefined fuzzy membership functions. The application domain is mobile robotics–a simulated robot performs obstacle avoidance tasks. The competence of the FCS controller was greater than what can be achieved in hand-coded fuzzy controllers.

In recent years, several studies have applied Michigan-style LCS to develop robot control systems. The robot navigation system in [11] learns to approach a light source while avoiding collisions with obstacles. Since the controller is based on a Michigan-style LCS, the implementation is quite similar to the work described in Chapter 5 of this thesis. With the evolved controller, a *KheperaII* robot is submitted to a real environment. Although this study involves an interesting matter – the translation of mature controllers (based exclusively on computer simulations) from simulated environments to real environments – it has not addressed the inherited problems in the LCS that significantly affect performance, for instance, over-general rules in population. The creation and rule-discovery processes tend to generate some over-general rules that are undesirable when attempting to discriminate between many sensor inputs. Over-general rules may also cause oscillation in strength; the result is that these rules may be used too often when they produce the wrong action and not often enough when they predict correctly.

Another Michigan-style LCS-based controller was developed by Foster in [28]. The main objective of this study was to deal with the problem that adaptive agents often receive ambiguous and limited sensor data from the environment. In order to overcome perceptual limitations of the environment, LCS research takes several different approaches, including: learning from past experience, developing sequences of rules, and multiagent LCS. Instead of investigating the evolutionary mechanism of LCS, Foster focused on a combination of these different approaches to perform a number of functions internal to the robot. To examine the impact of the ideas presented on the performance of the robot, the LCS was simplified without using rule-discovery and the bucket brigade algorithms that might introduce a stochastic element unnecessary for current applications.

Moussi *et al.* [70] investigated a Michigan-style LCS using NNs as its classifiers and applied it to robot navigation problems. The aim of this study was to examine whether NNs that typically work as universal approximators can enrich the performance of an LCS. Each classifier is replaced by the weights of two NNs: an *evaluation* network and an *action* network. The *evaluation* network determines the degree of interest for a message in the message list while the *action* network posts a new message in the message list. Although

the navigation task in this study is simple, it presents some advances – the addition of NNs takes advantage of their universal approximator characteristics and robot controller is able to manipulate real numbers.

The study of LCS-based robotics is still in the primary phase. Almost all navigation tasks (obstacle avoidance, going around the box, light-seeking tasks) and simulated environments (T-maze, simple non-Markov, symmetrical pilaster environments) in the studies described are non-complex. In this thesis, several revised LCSs based upon Holland's original formulation are explored and implemented to obstacle avoidance tasks in more complex simulated environments. The simulation results are compared with the results reported in [71].

Chapter 3

Mobile Robotics

More than one million industrial robots were at work by the end of 2005 [81]. Mobile robotics is a field in which a robot possesses mobility with respect to a medium without human assistance. Mobile robots execute complex tasks while navigating in an unknown environment; therefore they require control systems that can accommodate the uncertainty of the environment. The uncertainty in robotics generally describes the lack of reliable environmental knowledge or predictability because of randomness; and the uncertainty typically arises from noise and variation in sensory inputs, such as light and sound. A mobile robot needs the following characteristics:

- mobility – the ability to move with respect to the environment;
- autonomy – the ability to operate without human interaction;
- intelligence – the capacity to acquire and apply knowledge about itself and the environment [115].

An industrial robot used in a manufacturing plant works in a structured environment; its actions are repeated with regard to permanently placed elements – for instance, it may move crates from one moving line to another. But autonomous mobile robots that operate in partially unknown and unpredictable environments are desirable [26]. It is more difficult to program a robot to perform a predefined action when one does not know in advance what instant reactions will be required by the various situations the robot might encounter. It is

necessary to make online modifications to robot behaviour to deal with dynamic factors of the environment.

Hardware design, wheel design, kinematics analysis, sensors and perception, localization, mapping, and robot control architecture are disciplines that have grown to address the problem of robot navigation under unpredictable conditions. The design of a robot navigation system is analogous to the design of a robot brain, and thus relies on the discovery of new learning algorithms.

3.1 Robot Navigation

Navigation can be informally defined as a process to safely travel to a desired destination. In the context of robot navigation, the robot's sensory inputs must be used to determine the position and to make decisions necessary to move towards the destination. A more formal definition of robot navigation [100] is:

navigation is the task of moving a robot from its current position to a goal, based on noisy and incomplete information and in consideration of obstacles and boundary conditions.

The navigation task can be further broken down to goal seeking and obstacle avoidance, which is the subject of this thesis. Goal seeking behaviour requires knowledge of the locations of robot and goal. In a structured (mapped) environment, path planning algorithms, such as A* [78] can be used to determine an optimal route to the goal. However, in the presence of unexpected obstacles or in unmapped environments, obstacle avoidance behaviour has much higher importance. When sophisticated obstacle avoidance behaviour is achieved, goal-orientation can be added by a suitable behaviour arbitration scheme, such as context dependent blending [93].

From another perspective, robot navigation can be described using five separate phases [9]: *measurement*, *mapping*, *perception*, *planning*, and *action*, as shown in Fig. 3.1 and described below. Navigation is subordinate to a higher-level process, task definition, which

specifies the destination (goal). The task planner also imposes constraints, such as traversal time, on the path constructed by the path planning process.

Measurement: occurs at the beginning of the navigation process, when the robot employs sensors to collect the information about the environment. The gathered sensory data will be used to help map the environment. The measurement task also provides feedback to other tasks about the current state of the environment.

Mapping: creates a model of the environment to be navigated by the robot. To efficiently carry out complex navigation in an indoor environment, an autonomous mobile robot must be able to acquire and maintain a model of the environment. Implementation of the mapping task requires a data structure that represents the environment efficiently, such as an array representing the environmental grid. During the course of navigation, the map of the environment can either be explicitly predefined or be built dynamically. The mapping task is difficult and far from being solved due to several factors imposed by practical limitations on a robot's ability to build models. The limiting factors include: perceptual limitations, sensor noise, complexity and dynamics of the environment, etc.

Perception: similar to the measurement task – sensors are used to gather information from the environment to support the learning process. By means of perception, the robot senses the environment, detects objects, then, in conjunction with the mapping task, calculates the distance driven.

Planning: an optimal path is chosen based on constraints imposed by the robot's mission parameters. If there is no predefined environment map, the planning process devises paths between objects using the model constructed by the mapping task. Because the planning and mapping tasks sometimes merge due to the lack of an environment model, the robot must embark on a dynamic mapping and path planning exercise.

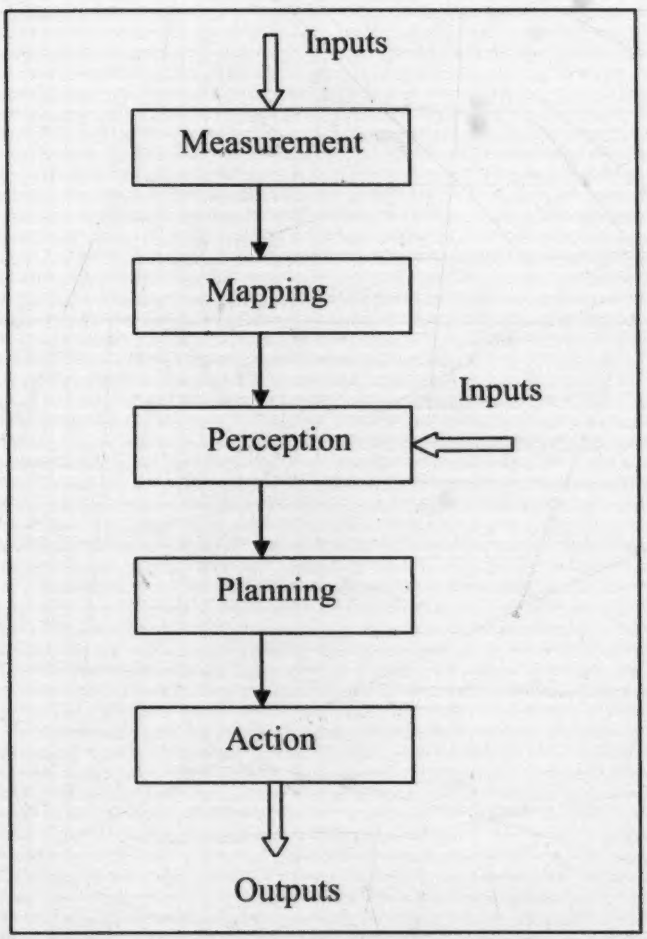

Fig. 3.1. The basic process of the robot navigation

Action: the process of driving over the path picked by the path planning task. The supervisory perception process monitors all incoming sensory data in order to follow the right path. The perception process compares sensory data with known obstacles in the existing environment map. When the perception task detects a possible collision, the action task initiates an appropriate evasive action. With recommendations from the perception task, the action task detects unmapped or newly discovered obstacles and initiates commands to avoid them.

In this thesis, computational intelligence techniques are applied to obstacle avoidance to imbue robots with learning ability. Therefore, unmapped environments are more suitable to test performance. The action and perception tasks together create a new path on-line, avoiding newly discovered obstacles.

3.2 Robot Sensing

A mobile robot is equipped with sensors that detect elements of the environment. The choice of one sensor over another depends on what task the robot needs to accomplish. In choosing a sensor, several considerations are: speed, reliability, physical constraints, cost, and preference, as well as available technology [14]. Mobile robot sensors can be *internal* or *external*.

- Internal sensors measure the intrinsic status of the mobile robot. The odometer is the most important internal sensor for an all-wheeled mobile robot. Odometric information is extracted by counting the number of rotations of axes involved in the motion of the robot. Position, speed, and acceleration all can be obtained by the means of odometry.
- External sensors are used to sense the environment in which the mobile robot operates. External sensors are usually more important than internal sensors since the primary function of a mobile robot is to interact with the environment. There are many types of external sensors: ultrasonic, range finders, vision and telemetry sensors, tactile sensors, etc. Most robots use multiple sensors in cooperation rather than

a single type of sensor.

Range Finders

Range finders are the most common among external sensors and can be grouped into two categories [4]: *passive* and *active*.

- Passive range sensors detect data from existing energy such as reflected light in the environment. A video camera is a passive range finder capturing images for further computations. Charge-coupled cameras are becoming popular for more complicated tasks, especially to achieve mobility in 3-dimensional space. Processing time requirements are generally high in passive ranger finders [23], thus it is not practical to apply passive range sensing techniques exclusively for real-time robot navigation. Other sensors are often used to cooperate with passive range finders: e.g., sonar sensors for measuring distance and tactile sensors for protection purposes.
- Active range sensors project energy–for example, laser or sonar waves–into the environment and measure a reflected signal. Most active range finders use time-of-flight (TOF) to measure the distance of obstacles along certain directions. A signal is sent to the surface of an object, and its distance is calculated by measuring the time delay between sent and returned signals. The absolute range to an observed point is directly available without complicated processing and analysis as the returned signal follows essentially the same path as the sent signal. Range information is used to build maps, calculate positions, and avoid obstacles. Common active range finders include ultrasonic, laser, and infrared sensors. Since active range finders rely on the reflected signal, problems arise if the object surface is highly reflective or highly absorbent. In such cases little or no signal may be received. Painting object surfaces or employing high-powered signal devices may alleviate these problems, but modifications take time and cost money.

Sonar Sensor

The ActiveMedia Pioneer 2-DX robots used in this thesis are equipped with various types of external sensors. Sonars were chosen since they are safe, low cost, widely available, and easy to use.

Sonar is a TOF distance-measuring device, which measures time between transmission and reception of signals. Sonar is a "wide beam" sensor; that is, the ultra-sound sent out is usually in the shape of a cone. Most mobile robots have a ring configuration of sonars to allow complete radial coverage. Using a wide beam, the scanning area is better covered with less sensors, but the angular resolution is lower.

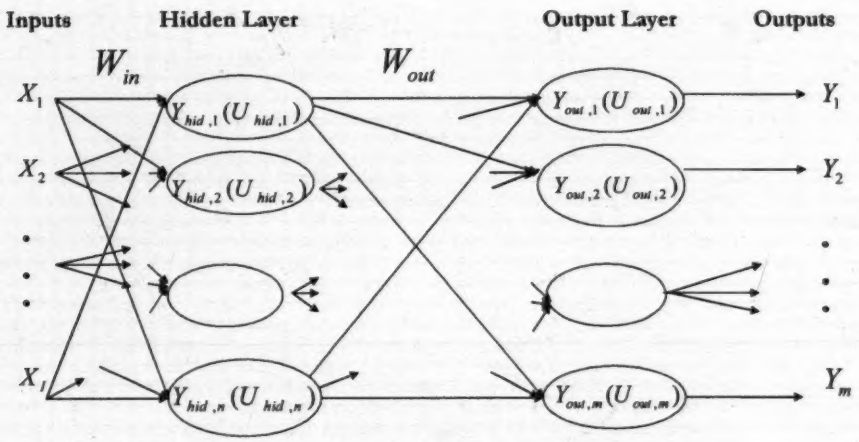

Fig. 3.2. The MLP robot controller architecture

3.3 Robot Controllers Based on Computational Intelligence

3.3.1 Neural Controllers

Robot navigation can be controlled by a MLP trained with a BP. The learning procedure of a perceptron-like NN is shown in Fig. 3.2. Training MLPs with BP results in nonlinear mapping or association structure. Therefore, with the input patterns derived from sensor readings and output patterns corresponding to control variables, the MLP will have its connection weights adjusted by the BP algorithm to develop a specific nonlinear mapping. To drive robot agents with trained neural controllers in real environments, robot sensor

readings are fed into NN inputs and the resulting network output values are applied to control robot actions, such as speed or heading.

The current sensor readings collected by the robot, X_1, X_2, ..., X_I, are augmented by a bias term X_0 and scaled by the input layer neuron weights, W_{in}, summed as U_{hid} and fed into each hidden neuron activation function, $Y_{hid}(U_{hid})$. The summations of weighted inputs are given by

$$U_{hid,n} = \sum_{i=0}^{I} (W_{in} X_i) \tag{3.1}$$

Similarly, outputs from hidden layer neurons, $Y_{hid,n}$, are scaled by the output neuron weights, W_{out}, summed as U_{out} and then fed into the output neurons

$$U_{out,m} = \sum_{n=0}^{N} (W_{out} Y_{hid,n}) \tag{3.2}$$

The NN is composed of neurons with a logistic activation function

$$Y(U) = \frac{1}{1 + e^{-U\alpha}} \tag{3.3}$$

where α is the slope parameter.

There can be an arbitrary number of hidden layer neurons; however, each output neuron produces a single effector control value, so the number of output neurons matches the number of effectors on the robot.

As previously mentioned, the BP algorithm is developed to train a MLP network. The gradient descent approach intends to minimize an energy function representing the instantaneous error defined as:

$$E = \frac{1}{2} \sum_{m=1}^{M} (D_m - Y_m)^2 \tag{3.4}$$

In this equation, D_m represents the m-th desired output pattern of the network. Using the steepest gradient descent approach, the learning rule in any network layer is given by:

$$\triangle W_{ji} = \frac{\partial E}{\partial W_{ji}} \tag{3.5}$$

Then δ_{out} is obtained as

$$\delta_{out} = (D_m - Y_{out,m}) \frac{\partial Y(U_{out})}{\partial U_{out}}, \tag{3.6}$$

and in the case of hidden layer,

$$\delta_{hidden} = \frac{\partial Y(U_{out})}{\partial U_{out}} \sum_{m=1}^{M} \delta_{out} W_{out,m,n} \tag{3.7}$$

The updated weights in any layer are then determined as

$$W_{ij}(k+1) = W_{ij}(k) + \mu \delta Y_i \tag{3.8}$$

where $\mu > 0$ is a learning rate.

All weights are initialized to small random values at the beginning of the training process. After a few iterations of training, the NN controller with a modified weights set is put into the same environment to test its performance [95].

Although NN controllers have been successfully implemented on real vehicles, there are several inherent weaknesses in this approach [19, 20]. First, the training is off-line, and the training data is usually collected from a specific class of path. In the case of a path type change, an arbitration between different responses is necessary. If the number of inputs is small, the MLP network can be trained (or retrained) quickly. However, for large input spaces, training is time consuming and the performance of the BP algorithm decreases [94].

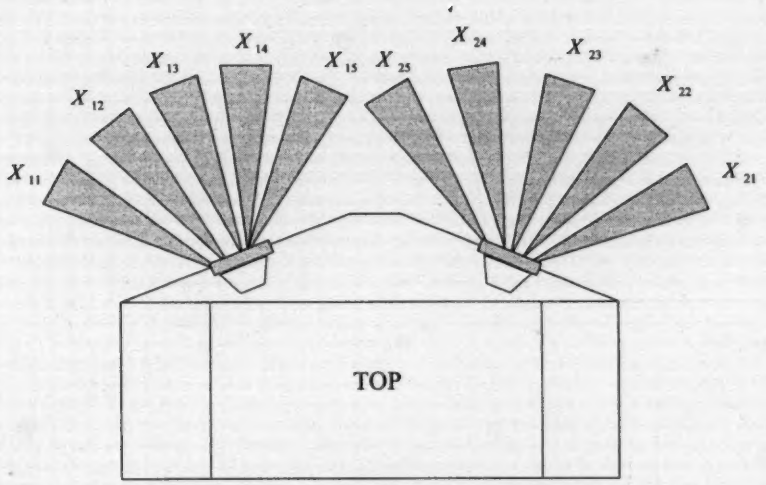

Fig. 3.3. The configuration of sonars

This fact can compromise the control of robotic systems where there are, in most cases, too many variables to consider.

Researchers have begun to investigate the reactive navigation system capable of online learning using trial and error methods. Grossberg *et al.* [10,31,32] developed a large collection of specialized NNs characterized by their focus on self-organization and unsupervised learning ability that are especially promising for applications in mobile robotics.

3.3.2 Fuzzy Controllers

The model introduced here is essentially the same as the fuzzy controller in [105] except that asymmetrical triangular functions are used instead of Gaussian functions. Triangular functions are suitable for fast computation–essential under real-time conditions. The following steps describe the operation of the FC:

(1) Read sonar data and construct membership functions for input:

All data from 10 sonar sensors (five front-left; five front-right) are received. The sonar configuration is shown in Fig. 3.3.

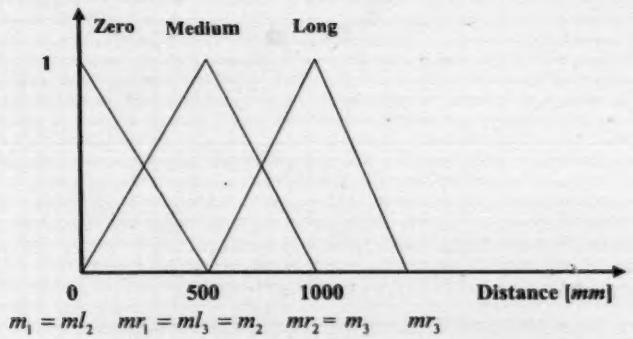

Fig. 3.4. Input membership functions for each input

Three input membership functions are constructed, that is, zero distance (0 mm), medium distance (500 mm), and long distance (1000 mm) as shown in Fig. 3.4. Each membership function is considered in a triangular form given by:

$$\mu_{F_i}(x_i) = \begin{cases} max(\frac{(x_i - ml_i)}{(m_i - ml_i)}, 0) & \text{if } x_i < m_i \\ 1 & \text{if } x_i = m_i \\ max(\frac{(x_i - mr_i)}{(m_i - mr_i)}, 0) & \text{if } x_i > m_i \end{cases} \tag{3.9}$$

where ml_i and mr_i are the x-coordinates of the left and right zero crossings respectively, and m_i is the x-coordinate of the peak of the triangle.

(2) Construct membership functions for output:

Five output membership functions are constructed for robot angular velocity ω: positive (30), small-positive (15), zero (0), small-negative (-15), and negative (-30). The triangular function of output is described by Eq. 3.9.

(3) Construct rules for obstacle avoidance:

Since the fuzzy inputs are the fuzzy sets from 5 front-left and 5 front-right sonar sensors, five fuzzy controllers could be constructed for obstacle avoidance behaviour, and thus five rule bases can be created using a pair of symmetrical sonars from both sides. Each rule

ω	X_{2i} [mm]		
X_{1i} [mm]	**0**	**500**	**1000**
0	0	-15	-30
500	15	0	-15
1000	30	15	0

TABLE 3.1

Rules for avoiding frontal obstacles

base depends on the sensor arrangement. The rules for avoiding frontal obstacles using X_{11} and X_{21} are shown in Table 3.1.

The results from Table 3.2 are listed as follows:

If X_{11} = 0*mm* and X_{21} = 0*mm*, *then* ω = 0

If X_{11} = 0*mm* and X_{21} = 500*mm*, *then* ω = -15

If X_{11} = 0*mm* and X_{21} = 1000*mm*, *then* ω = -30

If X_{11} = 500*mm* and X_{21} = 0*mm*, *then* ω = 15

If X_{11} = 500*mm* and X_{21} = 500*mm*, *then* ω = 0

If X_{11} = 500*mm* and X_{21} = 1000*mm*, *then* ω = 15

If X_{11} = 1000*mm* and X_{21} = 0*mm*, *then* ω = 30

If X_{11} = 1000*mm* and X_{21} = 500*mm*, *then* ω = 15

If X_{11} = 1000*mm* and X_{21} = 1000*mm*, *then* ω = 0

In the case that both X_{11} and X_{21} are 0mm, the robot might be approaching a symmetrical corner. FC will not send rotation output but will drive the robot backward (emergency behaviour [105]).

To calculate the fire strength of each rule, a *t*-norm product operator is used. A *MAX_MIN*

inference method is then used to obtain the output membership function strengths.

(4) Defuzzification:

In this step, membership values are defuzzified for the control action of angular velocity using a defuzzification method. The defuzzification of data into a crisp output is accomplished by combining the results of the inference process and determining the fuzzy centroid of the area. The output y_k for angular velocity is then given by

$$y_k = \frac{\sum_{i=1}^{n} \mu_A(y_i) y_i}{\sum_{i=1}^{n} \mu_A(y_i)}, \tag{3.10}$$

where $\mu_A(y_i)$ is the output membership function strength, n is the number of discrete points defined on the output fuzzy sets, and y_i is the i-th domain point.

This simple fuzzy controller can solve complicated situations, allowing the robot to avoid all obstacles detected by sonar sensors [105]. It is usually not necessary to evaluate every possible input combination since some of them may rarely or never take place; thus, the experienced designer can reduce the size of the rule-base, simplifying the process and shortening computing time.

Although fuzzy controllers are simple to design and faster to prototype than conventional control systems, they require more simulation and fine-tuning before they are usable. In addition, the design of the rule base depends heavily on the experience of the experts during the construction of rule base.

3.3.3 Evolutionary Robot Controllers

A NN-based robot controller can be trained with a GA [76]. First, a simple controller with prior knowledge is formulated to drive robot navigation in a maze world without running into obstacles or getting stuck in the arena. The controller is designed with a set of simple rules that map sensor inputs to wheel motor outputs so that the motion of the robot is a

direct function of current sensor inputs. The rules are defined as follows:

- *Rule 1*: Each wheel motor speed is set proportionally to the sum of the range-finding sensor readings on the opposite sides of the robot.
- *Rule 2*: If the total sum of all the range-finding sensor readings is less than a predefined threshold, then wheel motors on both sides are given a negative speed command.

Rule 1 causes the robot to veer away from objects or to remain centered between the walls of corridors, while *Rule 2* enables the robot to escape from obstacles or corners.

When this knowledge-based robot is running, all robot sensor inputs and corresponding motor outputs (wheel motor speeds) are collected at each instant. The resulting data set is then used to train an NN using EC. The structure of the NN is the same as the MLP network described in Section 3.3.1, however it is trained by a GA instead of a BP algorithm. The connection weights of the NN are represented by a set of real-valued numbers. In terms of the notation introduced in Section 3.3.1, the genotype structure c is defined as:

$$c = [W_{in_11}, ..., W_{in_1N}, ...W_{in_IN}; W_{out_11}, ..., W_{out_1M}, ..., W_{out_NM}] \tag{3.11}$$

At the beginning of the training, all weights of the NN are initialized randomly to small real values. A two-point crossover and mutation are applied to c at each iteration to make a new genotype, c'. Fitness proportional selection is executed before the application of crossover and mutation. The fitness function is defined as the mean-square-error (MSE) similar to the energy function E in Eq. 3.4:

$$f(j) = \frac{1}{2}\sum_{m=1}^{M}(D_m(x_j) - Y_{out,m}(x_j))^2, \tag{3.12}$$

where $D_m(x_j)$ is the desired output of the m-th output node when it is presented with the j-th set of training input x_j and $Y_{out,m}(x_j)$ is the actual output of the m-th output node during training. The performance of the NN with weights c' is then compared to that of the

network with weights c. If the performance is better (e.g. f becomes smaller), c' replaces c; otherwise c remains in the population and the process is reiterated.

The values of crossover probability P_c and mutation probability P_m are important for the algorithm. They are adaptively adjusted at each generation depending on the quality of the current solution. The probabilities could be high in the early stage of the training when the solutions are poor. As the training proceeds, the solutions are improved, and the probabilities decrease simultaneously. The probabilities are calculated as

$$P_{m,c} = P_{m_base,c-base} + H\frac{f_{current}}{f_{base}} \tag{3.13}$$

where $f_{current}$ is the current fitness obtained by Eq. 3.12 and f_{base} is set as the fitness calculated at the first iteration of training. H is a scaling factor typically close to unity that governs the extent to which the probabilities are influenced by training errors. The base (minimum) probabilities, P_{m_base} and P_{c_base}, are given by

$$P_{c_base} = \frac{1}{N_p}, \tag{3.14}$$

$$P_{m_base} = \frac{1}{N_p L}, \tag{3.15}$$

where N_p is the population size and L is the length of the genotype.

Maintaining the probabilities of genetic operators changing allows initial poor solutions to move more quickly through the solution space while not disturbing good solutions in the final stages of training. After training, the resulting NN controller is used to control the robot in similar environments to test its performance. According to the simulation results, such an evolutionary controller can produce successful obstacle avoidances, exhibiting features the same as the original knowledge-based controller [76].

Although ER technology has developed very quickly in the last few years, new theoretical and empirical tools are needed to develop a practical technology [99]. In particular, the dynamics of evolution in ER search space needs more work. Also, more research is needed on properties of the underlying solution base of robot controllers.

3.4 Issues and challenges

In this thesis, EC techniques are used to learn obstacle avoidance behaviour. The major goals of this study are:

- Application of ML algorithms to general and sophisticated behaviours: Obstacle avoidance behaviour is examined because it is considered the benchmark test for mobile robotics. More complex navigation behaviours can be developed by increasing the complexity and dynamics of the environment, then comparing performance with hand coded techniques.
- Coupling of simulation to reality: Using actual robot interaction with the real world as the only form of performance evaluation is too slow and expensive. For this reason, computer simulation is a good choice for implementation. But, how to transfer the successful controller evolved in simulation to a real robot is always an issue. In this study, a high quality simulator giving virtual replacement for a real robot is employed. Nevertheless, extra noise caused by the environment needs to be added to make the simulations closer to reality.
- Controller architecture: An LCS is adopted as the basic architecture of the robot controller because it learns syntactic rules consisting of strings of 1s and 0s–a binary structure easily implemented by a computer. In addition, the majority of current ER work uses NN-based controllers with simple network topologies and restricted weight values. In this thesis an LCS-like NN is adopted as the controller structure with a GA used as the primary learning algorithm to update the connection weights. Different parameters of GA are explored to improve robot learning ability and robustness of evolved controllers. In particular, population size and probabilities of

GA operators are examined.

Challenges for controllers that navigate robots in complex indoor environments include:

- *Autonomy*: There is need of an autonomous system that incorporates the whole logic for navigation, without using a remote control system. In some cases, autonomy also implies that the robot is equipped with a sufficiently long power supply.
- *Complexity and dynamics*: Besides static obstacles (fixed objects such as walls), a robot might encounter dynamic obstacles (moving objects or environments changed by other agents). In such environments, a static map cannot be relied on during navigation.
- *Sensor noise*: Uncertainties in measurement usually spoil sensory data. Noise in sensory input with unknown distribution is a source of uncertainty. This noise is more intense in a real robot than in a simulated one.
- *Drift/slippage*: Slippage of robot wheels causes motion inaccuracy, for instance, departure from the correct path. At the same time, odometric errors accumulate with time, so even the smallest rotational error may produce a huge effect on subsequent translational error in locating the robot's position.
- *Real-time requirements*: Robots operate continuously, so environment perception must be simple and easily accessible. Therefore, the choice of sensors used in measurement/perception tasks should be appropriate for robot speed, sensor reaction time, etc.

Chapter 4

Learning Classifier Systems

To survive in its environment, an animal's brain must make associations between environmental messages and consequent actions that lead to satisfaction of its needs [114]. Some of these associations are possessed by the animal at birth, but the rest must be learned through experience. An autonomous mobile robot controlled by a ML system is in a similar situation. Since the notion of ML occurred to humans, many learning systems have been developed. The LCS, first introduced by Holland and Rietman [43], possesses the salient properties necessary for learning. It is a learning system using a GA to modify its behaviour in a given environment.

In this chapter, principles of ML are introduced, followed by an overview of GAs. The LCS, central to this thesis, is introduced and a variant, XCS, is presented for comparison.

4.1 Machine Learning

4.1.1 Definitions

ML is concerned with the construction of computer programs that allow computers to learn automatically to improve with experience. Mitchell gives a more precise definition in [68]:

A computer program is said to **learn** from experience E with respect to some class of

tasks T and performance measure P, if its performance at tasks T, as measured by P, improves with experience E.

Let T be the task of a robot travelling from one office to another. Performance measure P is the average distance travelled before an error (for example, a collision as judged by a human overseer) occurs. Training experience E is recorded as a sequence of images and steering commands during the travelling period.

Learning is defined by Narendra and Thathachar [75] as "... the ability of systems to improve their responses based on past experience." The "systems" mentioned here can be biological or artificial (computer). A more technical definition is provided in [75]: "... the term 'learning automation' describes both deterministic and stochastic schemes used in discrete systems that improve their performance in random environments." In [66], Michalski *et al.* give a comprehensive definition from a cognitive viewpoint: "Learning is a many-faceted phenomenon. Learning processes include the acquisition of new declarative knowledge, the development of motor and cognitive skills through instruction or practice, the organization of new knowledge into general, effective representations, and the discovery of new facts and theories through observation and experimentation." These definitions commonly describe learning as a process to improve behaviour toward the environment.

Behaviour is an outcome of the interaction between an entity (biological or mechanical) and its environment. The behaviours of an autonomous mobile robot are therefore determined by the structure and dynamics of both the robot and its environment, in particular by the interface between them. ML categorizes two types of behaviours [91], S-R and *dynamic*. S-R behaviour is a reactive behaviour; that is, a stimulus results in a direct response. On the other hand, dynamic behaviour contains an internal state that launches a complex processing procedure before a response is generated.

4.1.2 Objectives

Current ML research objectives can be classified into three areas [66]:

1. **Task-oriented study**: intends to design new learning systems to carry out a set of predefined tasks.
2. **Cognitive simulation**: aims to explore the dynamics of human learning behaviours and utilizes the computer to simulate human learning processes.
3. **Theoretical analysis**: exclusively discovers novel learning algorithms in the theoretical domain.

Although ML distinguishes these three different objectives, progress toward one objective usually promotes the development of others. Instructing a computer or a computer-controlled robot to perform a task leads to various applications. Task-oriented study is more practical and therefore it is the focus of this thesis.

4.1.3 Issues

Environmental information is incoming at each instant, in arbitrary order, interspersed with dynamic instances. Typically, a large amount of information should be stored over time and processed later, however, it is more realistic to assume that the learning system has a limited storage capacity and extracts information the instant it occurs. In such a situation, the entity will be more likely to learn from its environment rather than its internal model. This is similar to the assumption that an animal learns incrementally, with essentially no direct memory of events in their original form or detail [114].

In the real world there are generally no teacher-like environments to tell the entity what was right, or to provide error feedback. It is possible to rate high the action which brings immediate satisfaction of the present requirement. Although "degree of satisfaction" provides a measurement of current action, it doesn't guarantee that other actions would not have resulted in better satisfactions. In [114] the term "payoff" is defined as "degree of need satisfaction". Receiving payoff implies that the learning system is gaining some degree of need satisfaction as a consequence of interacting with the environment in a particular way.

Problems arise if payoff is uniquely used, since in most occasions, no payoff at all is received after an action. A certain action, for example, causes the entity to move to a new

location so that the next action will result in a large payoff. Such an action chain could be long or short. To evaluate the indirect actions promoting the last payoff, proper credit should be associated with intermediate steps.

In terms of the issues mentioned above, robot learning is analogous to the *animate* problem technically defined by Wilson [114]: "... incremental learning of multiple disjunctive concepts under payoff. To the extent this is an accurate characterization of the organism's learning problem, simulations that succeed in emulating an organism's behaviour in realistic environments should, in their computational principles, suggest models of the organisms themselves, which can be compared with models from other sources." Wilson concludes that the design principles for autonomous robots can benefit from how natural organisms learn incrementally without storing raw experience.

4.1.4 Approaches

Many techniques have been proposed in ML. Michalski [66, 67] classifies several main research paradigms: knowledge-intensive and direct implementation learning, learning from instruction, and learning from examples.

Since the autonomous robot in general does not have prior knowledge of the environment, the first type of paradigm is the least interesting way of learning. With a large amount of built-in domain knowledge, the system itself does little except programming or storing the information in a database.

Learning from instruction is like receiving education at school. The system needs to understand the instructions first, and then translate them into internal knowledge and store this knowledge. Finally, the system integrates the new knowledge with what it already knows. As there are no teacher-like environments in the real world, this type of paradigm is not directly of interest.

Learning from examples is representative. Examples from the environment are presented to the system. Simply stated, what the system intends to learn can be considered as the functions (typically nonlinear) mapped from inputs to outputs. Information that in-

dicates the influence of example is associated with that example. With respect to each response of the system, the feedback from the environment will indicate whether this response was good or bad. If the information is provided after the system has produced a response, the system can be considered a RL system. A formal definition of RL is given below.

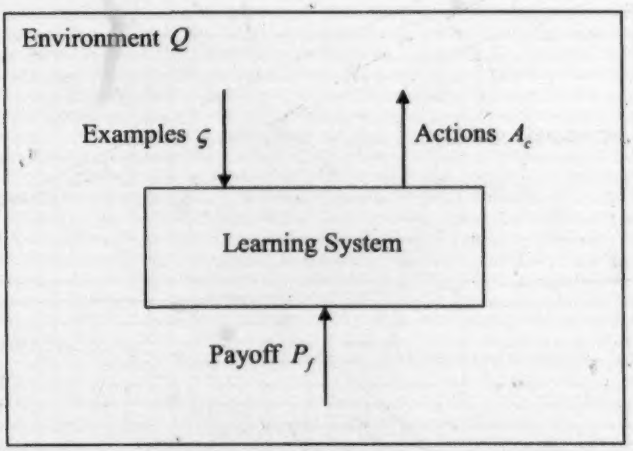

Fig. 4.1. The process of reinforcement learning

As indicated in Fig. 4.1, two sets are defined such that set Q represents the states of environment and set P_f represents responses of the learning system that are executed in the environment. The learning system is then described as a feasible function with respect to time instant t,

$$\varphi = \zeta(\varsigma, t) \tag{4.1}$$

in which $\varphi \in A_c$ and $\varsigma \in Q$. Feedback from the environment is defined by:

$$P_f = \phi(\varphi, \varsigma, t) \tag{4.2}$$

where $P_f \in R^+$ is the payoff from the environment. The higher P_f, the better is the performance of the action. A system learns if the following inequality is true:

$$\phi(\zeta(\varsigma, t), \varsigma, t) > \phi(\zeta(\varsigma_{-1}, t-1), \varsigma_{-1}, t-1) \tag{4.3}$$

Although this definition is quite simple and specific to S-R systems, it is sufficient to describe a ML system in this thesis.

An LCS can be considered a system learning from examples which learns disjunctive concepts incrementally under payoff. From a functional standpoint, three algorithms are employed by an LCS [114]:

- Performance algorithm: determines short-term behaviour;
- RL algorithm: determines how the payoff affects the strength of each classifier;
- Discovery algorithm: uses a standard GA to reproduce offspring among high-strength classifiers.

Since the GA is a key component of an LCS, a detailed introduction is given in the following section.

4.2 Genetic Algorithms

4.2.1 History

The foundation of a GA can be traced to adaptive systems originated by Holland in the 1960s. Holland subsequently proposed the operators of crossover and other recombinations. Based on the previous works, the concept of adaptive digital systems using mutation, selection, and crossover is systematically presented in [40]. In [40,41], Holland illustrated

a GA from a theoretical point of view, indicating the essence that it simulates the process of biological evolution as a problem-solving strategy.

In the 1980s, GAs were applied to a broad range of domains, from abstract mathematical problems like graph coloring to engineering issues such as pipeline flow control, pattern recognition and classification, and structural optimization [30]. The reasons for their success are broad applicability, ease of use, and global search perspective.

Initially, the applications of GAs were simple. However, GAs are gradually moving into the commercial field. The development of GAs has expanded due to the exponential growth of computer speed. Nowadays, EC is thriving, and GAs are solving countless problems in diverse areas. Since the power of evolution has reached virtually every field, new uses for GAs continue to be discovered.

4.2.2 Fundamentals and Working Principles

A GA is a search algorithm based on the mechanism of selection and natural elimination. Goldberg explains GAs in the following way [30]: " ... They combine survival of the fittest among string structures with a structured yet randomized information exchange to form a search algorithm with some of the innovative flair of human search. In every generation, a new set of artificial creatures (strings) is created using bits and pieces of the fittest of the old; an occasional new part is tried for good measure. While randomized, GAs are no simple random walk. They efficiently exploit historical information to speculate on new search points with expected improved performance".

Fig. 4.2 is a flowchart of the working principles of a GA. Unlike classical search and optimization methods that pick only one solution, a GA begins its search with a set of randomly encoded solutions, namely, chromosomes or strings describing candidate solutions to the problem. The encoding of chromosomes depends mainly on the problem to be solved, typically representing the solutions as binary strings. After a population of solutions is created, each solution is evaluated and assigned a fitness value. The evaluation of a solution means calculating the objective function value including constraints. Thereafter, a

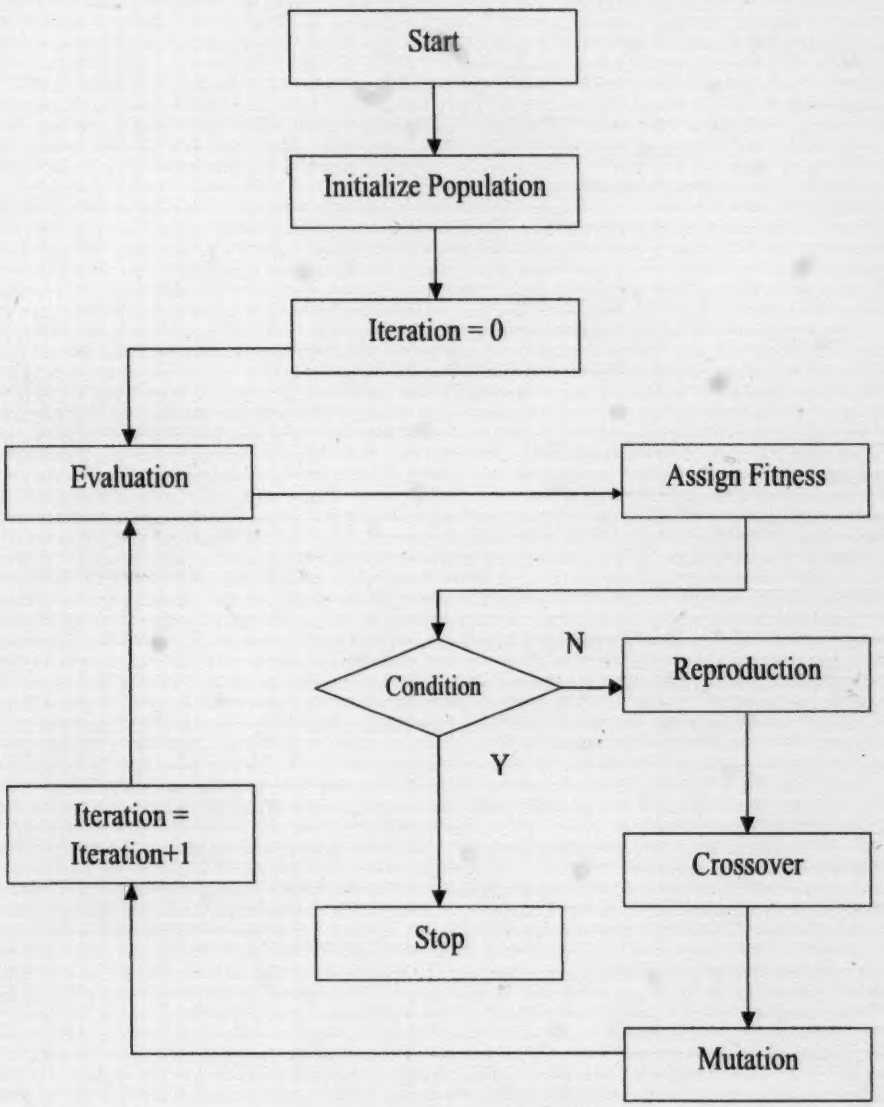

Fig. 4.2. Flowchart for a GA

termination condition is checked. If the termination criteria is not satisfied, the population is modified by three main genetic operators–*reproduction*, *crossover*, and *mutation*. The selection/reproduction operator selects high-performing strings, while the crossover operator recombines substrings from two such strings in the hope of forming a better solution. The mutation operator alters strings locally with the same goal. Since these operations are not performed deterministically, the goals are not guaranteed. However, it is expected that low-performing solution strings will be eliminated by the reproduction in subsequent generations while high-performing strings will be enhanced.

Solution Encoding

The encoded chromosome reflects the solution it represents. The representation scheme has a major impact on the performance of the GA. Representation schemes affect performances in terms of accuracy and computation times. The most often used encoding is a binary string in which every bit takes a value of either 0 or 1. An example of chromosomes is shown in Fig. 4.3. Each chromosome corresponds to one binary string and each bit represents a certain characteristic of the solution. To solve a specific problem, it is sometimes more appropriate to encode the solutions using integers of real numbers. New genetic encoding techniques such as bias code, uniform code, tri-sector code, and symmetric code can be applied for complex optimizations or robotic problems. These codes have found applications in developing special and powerful GAs [64].

Chromosome 1	1101100100110110
Chromosome 2	1101111000011110

Fig. 4.3. Examples of chromosomes

When a binary representation scheme is employed, it is important to determine the number of bits in each chromosome. Too few or too many bits can affect the performance adversely. A string must be encoded in such a way that it can be broken apart and combined with other strings to form new strings, designated as *offspring*.

Creation of Initial Population

At the beginning of processing, the GA requires a group of initial solutions. There are two methods to form the initial population. With the first method, the initial population is created by a random number generator and applied to problems that don't need prior knowledge. The second method employs prior knowledge about the given problems. Using this knowledge, a set of requirements is obtained and solutions which satisfy those requirements are collected to form the initial population. With the second method, the GA starts the evolution with a set of approximately known solutions and converges to an optimal solution; this takes less time than with the first method.

Evaluation and Assigning Fitness

After choosing a string representation scheme and creating the initial population, the next step is to evaluate these solutions and assign a measure of *goodness* to each solution. If there are no constraints, the fitness of each string is assigned a value corresponding to the solution's objective function. In most cases, the fitness is made equal to the objective function value. In other cases, it can be scaled, normalized, or inverted, depending on the particular problem at hand.

Selection or Reproduction Operator

The selection operator aims to make duplicates of good solutions and eliminate bad solutions in a population, keeping the population size constant. This is achieved by:

1. Distinguishing a good solution in a population;
2. Making multiple copies of good solutions;
3. Eliminating bad solutions from the population to make room for the copies of good solutions.

There are a number of ways to achieve the above tasks. The common methods are *tournament* selection, *proportionate* selection and *ranking* selection [30]. Proportionate selection is of particular interest in this thesis since parents are selected according to their fitness.

The higher the fitness, the higher the chance to be selected.

The implementation of a proportionate selection operator can be thought of as a *roulette wheel* mechanism. A virtual spinning wheel is divided into N (population size) sectors, each sector corresponding to a solution. The size of the i-th sector is proportional to the selection probability P_i as follows:

$$P_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{4.4}$$

where f_i is the fitness. For each individual, the cumulative probability is calculated:

$$P_{cum,i} = \sum_{k=1}^{i} P_k \tag{4.5}$$

The wheel is turned N times. At each turn, a random number g is generated in the interval [0, 1] with a uniform probability distribution. The following selection rule is then applied (g_0=0):

If $P_{cum,i-1} < g \leq P_{cum,i}$, then solution i is selected.

After N turns, a new population of the same site (N) is created.

Crossover Operator

In most crossover operations, two strings are randomly selected from the mating pool and portions of the strings are exchanged to create two new strings. In a single-point crossover operator (see Fig. 4.4) this is performed by randomly choosing a crossing site along the string and exchanging all bits on the right side of the crossing site.

The chance of creating better solutions is far better than random since the parent strings being crossed are not two arbitrary strings but strings that have survived from the earlier selection phase. In order to preserve good strings selected during the reproduction phase, not

Parent 1: **010** | **0001010**	Offspring 1: **010** | 1000110
Parent 2: 011 | 1000110	Offspring 2: 011 | **0001010**

Fig. 4.4. Single-point crossover operator

all strings in the population are used in a crossover. If a crossover probability, P_c, is used, then $100P_c\%$ strings are used in the crossover operation and $100(1- P_c)\%$ of the strings are copied to the new population. The idea of exchanging partial information between two strings can also be achieved with more than one cross-site (n-point crossover).

Mutation Operator

After the crossover is completed, mutation takes place. All individuals in the population are checked bit by bit, and the bit values are randomly reversed with a specified mutation probability P_m. Unlike crossover, one offspring is generated from a single parent string. The mutation operator forces the algorithm to search new areas and maintain diversity of the population.

Original offspring:	1100 **0** 10 **1** 110
New offspring:	1100 **1** 10 **0** 110

Fig. 4.5. Bit mutation operator

Fig. 4.5 shows how a string is mutated to a new string. In this example, the fifth and eighth bits are reversed. Quite often, the solution obtained after mutation is better than the original solution. Although this may not happen in all the instances of mutation operations, mutating a string with a small probability is not random since the process has a bias for creating only a few new solutions in the search space.

4.2.3 Parameters of Genetic Algorithms

There are three main types of control parameters in a simple GA; population size N, crossover probability P_c, and mutation probability P_m. Many researchers have investigated the effects of these parameters on the performance of a GA. Their conclusions can be summarized as follows.

N is the number of chromosomes in the population: a large N implies the simultaneous handling of many solutions and increases the computation time per iteration; accordingly, overall searching time is longer. However, since many samples from the search space are used, the probability of acquiring an optimal solution is higher than in cases that employ a small N. With a small N, a GA has a few possibilities to perform crossover and only a small portion of search space is explored; thus the search converges too quickly, often yielding only locally optimal solutions. A practical N is suggested to be between 50-200 [16], but this depends on the problem at hand.

P_c determines the frequency of the crossover operation. It is useful at the start of an optimization to discover a promising region. A low crossover frequency decreases the speed of convergence to such an area. On the other hand, if the frequency is too high, it can lead to saturation around a local minimum.

P_m is the frequency at which parts of the chromosome are mutated. A high mutation rate introduces high diversity in the population and might cause instability. On the other hand, absence of mutation is associated with poor performance caused by insufficient diversity of the search space.

4.3 Operation of Learning Classifier Systems

The LCS learns simple string rules called classifiers [42, 43]. The system assimilates a perpetual stream of information about the environment and creates a set of competing rules without significantly disturbing those already obtained. To accomplish a learning task, an LCS receives information about the environment, performs internal processing, then

provides an action which has a perceived effect on the environment. Feedback from the effects the system has had on the environment is used to modify the LCS's internal structure and parameters; the system effectively learns from experience.

The theory of LCS is fairly young. So far, two main approaches have been developed: the *Michigan approach* [43] and the *Pittsburgh approach* [17]. In the Michigan approach, an individual of an evolutionary algorithm population encodes a single rule, whereas in the Pittsburgh approach each individual represents an entire set of rules for the problem at hand [21]. This thesis employs the more developed Michigan approach.

From the structural point of view, a LCS consists of four parts [6] in the form of a quadruple (C, M, D, E):

1. **The input interface** D translates the current state of environment into standard format, typically called *detector*.
2. **The classifiers** C are the string rules used by the system to process information produced by the detectors.
3. **The message list** M contains all current messages, either from the environment or the actions from other classifiers.
4. **The output interface** E translates some messages into effector actions that will change the state of the environment.

The execution cycle of an LCS consists of three major components: a *rule and message subsystem*, an *apportionment of credit subsystem*, and a *rule discovery mechanism* [91]. Fig. 4.6 shows a block diagram of an LCS.

4.3.1 Rule and Message Subsystem

Detectors collect information from the environment and codify it into messages that the classifier systems can recognize. In the simplest version, all messages are of a fixed length in a specific alphabet (typically k-bit binary strings [91]).

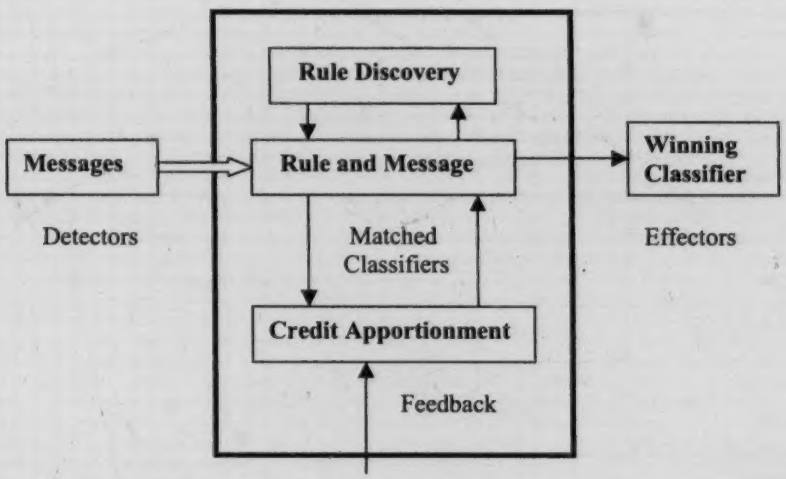

Fig. 4.6. Block diagram and data flow of a typical LCS

Each classifier in the classifier population consists of a rule in the form of a triple (λ, φ, S):

- λ is a string representing one or more conditions as the antecedent, drawn from the ternary alphabet (0, 1, #). The symbol # in the condition means "don't care" and can assume a value of either 0 or 1. The more "don't cares," the more general the rule. The number of non-# symbols in the antecedent, relative to its length, represents the *specificity* q^{λ} of a classifier rule;
- φ is a string representing an action part as the consequent, composed of a binary string (0, 1). The length of the action part depends on the problem being solved.
- S is an associated strength of each classifier. The strength part of a rule provides a measure of its past performance in the environment in which it is learning: the higher the strength, the better the rule's performance and the more likely it is to be used when its conditions match an environmental message.

After a message from the environment is encoded as M_x, the conditions of all classifier rules in the population C^{λ} are matched against the message. A matching function F_m is in

the form of a product operator:

$$F_m(X,Y) = \prod_{i=1}^{L} f_m(x_i, y_i), \tag{4.6}$$

where L is the length of the condition, x_i and y_i are the i-th bits of string X and Y, respectively. The bit match function f_m is defined as

$$f_m(x_i, y_i) = \begin{cases} true, & \text{if } x_i = y_i \\ \\ false, & \text{otherwise} \end{cases} \tag{4.7}$$

For the j-th classifier c in the population:

$$\text{if} \begin{cases} F_m(M_x, C_j^{\lambda}) = true, & c \text{ matches the message } x \\ \\ F_m(M_x, C_j^{\lambda}) = false, & c \text{ does not match the message } x \end{cases} \tag{4.8}$$

If the entries at all non-# positions are identical, the condition matches the message. For example, the encoded binary environmental message:

1 1 1 0 0 0 1 1 1 0

can match all the following classifiers:

Conditional part λ	Action part φ
1 1 1 0 0 0 1 1 1 0	0011
# 1 1 0 # 0 1 1 # 0	1001
1 # # 0 # 0 # # 1 0	0000
# # # # # # # # # #	1111

Every rule in the population that matches the environmental message is then submitted to the apportionment of credit subsystem. Since classifiers only match environmental

messages and actions only take effect on the environment in this thesis, this LCS is an S-R system.

4.3.2 Apportionment of Credit Subsystem

The apportionment of credit subsystem is responsible for modification of rule strengths as the system learns [91]. Modification occurs between two time steps of the process, and must be completed before a new input is fed into the system. This task is difficult since overt rewards are rare in complex environments. The *bucket brigade* algorithm is designed to solve the apportionment of credit problem. This algorithm adjusts the strength to exhibit the classifier's overall usefulness to the system. The strength is then employed as a fundamental competition. The bucket brigade algorithm has two main components: *auction* and *taxation*.

All classifier rules that match the environmental message compete in an auction to determine the individual that will have an effect on the environment. The bucket brigade algorithm allows the strongest classifiers to post their messages. Since the system is an S-R system, only one classifier will eventually win the auction to modify the environment. A classifier enters the auction with a bid, which is a function of its strength and specificity. The bid of rule i at iteration t, $B_i(t)$, is calculated as:

$$B_i(t) = \rho S_i(t) q_i^{\lambda} \tag{4.9}$$

where ρ is a parameter that acts as an overall risk factor influencing what proportion of a rule's strength will be bid (and possibly lost) in a single step, $0 < \rho \leq 1$; $S_i(t)$ is the current rule strength; and q_i^{λ} is the specificity, $0 \leq q_i^{\lambda} \leq 1$. Eq. 4.9 indicates that highly specialized rules with values of q_i^{λ} close to 1 are preferred over general rules.

After all bids for rules in the matching pool are calculated, the winner of the auction is determined by selecting the individual with the highest bid. As in a real auction, the winner pays for the privilege to perform its action by decreasing its strength by the amount of its

bid.

$$S_i(t+1) = S_i(t) - B_i(t) \tag{4.10}$$

Other classifier rules that participated in the auction but did not win save their bids and their strength is not affected.

Two types of taxes, Tax_{life} and Tax_{bid}, are used to prevent the classifier population from being cluttered with individuals of little or no utility having artificially high strengths. For individuals that do not match a message in the current iteration, a fixed rate tax Tax_{life} is levied. Suppose a rule is inactive for n consecutive iterations. Its strength is modified according to:

$$S_i(t+n) = S_i(t)(1 - \text{Tax}_{\text{life}})^n \tag{4.11}$$

For rules that do not win the auction but are in the current matching pool, a fixed rate tax Tax_{bid} is applied to penalize their generality. Therefore, after n iterations of losing, the classifier rule has strength:

$$S_i(t+n) = S_i(t) - \sum_{j=0}^{n} \text{Tax}_{\text{bid}} B_i(t+j) \tag{4.12}$$

After the winning classifier is generated, it is necessary to determine whether the action was beneficial or detrimental to the environment. The learning process requires either positive or negative feedback from the environment as a consequence of an action. The feedback is then used by the apportionment of credit subsystem to appropriately reinforce or punish the winning rule by increasing or decreasing its strength. So the strength of the winning classifier rule $C_i(t)$, at the end of iteration t, is:

$$S_i(t+1) = S_i(t) + R_i(t) - B_i(t) \tag{4.13}$$

where $R_i(t)$ is the reward from the environment at iteration t. Because the action caused by the winning rule could be beneficial or detrimental, the reward $R_i(t)$ will correspondingly hold a positive or negative value.

Finally, the general strength update equation can be obtained by combining Eqs. 4.11-4.13:

$$S_i(t+1) = (1 - \text{Tax}_{\text{life}})S_i(t) + R_i(t) - \text{Tax}_{\text{bid}}B_i(t) \tag{4.14}$$

The update configuration of Eq. 4.14 is set as follows:

- For the winning rule, Tax_{life} is set to 0 and Tax_{bid} is set to 1;
- For non-winning rules in the matching pool, Tax_{life} is set to 0 and Tax_{bid} is set to a fixed rate of less than unity;
- For non-matching rules, Tax_{bid} is set to 0 and Tax_{life} is set to a fixed rate of less than unity;
- Reward $R_i(t)$ is received only by the winner and is equal to zero for the remaining rules;
- Bid $B_i(t)$ is determined by Eq. 4.9 for all those in the matching pool.

4.3.3 Rule Discovery Mechanism

After a certain number of iterations, a GA is applied to produce the next generation of rules. Based upon genetic operators, the search process is directed along trajectories influenced by the (above-average) strength of the current classifier population. The GA normally selects the rules with greater strength and promotes reproduction among them. The offspring generated are launched into the next generation to compete with their parents and other individuals. If the new classifiers are more effective in gaining payoff, they will survive and replace the weakest individuals in the population.

4.4 Extended Classifier System (XCS)

An advanced learning classifier system technique, XCS, is introduced for a comparison performed later in the thesis. The XCS, introduced by Wilson [113], has many features in common with his earlier work with ZCSs [112].

In a Michigan-style LCS, the strength of the classifier acts as a predictor of payoff as well as the fitness of the GA. However, the unique use of strength may not adequately describe fitness. Unlike an LCS, an XCS employs the accuracy of classifier reward prediction serving as the classifier fitness, which is an inverse function of the classifier's average prediction error. The strength is therefore replaced by three parameters: payoff prediction p, prediction error ε, and fitness F. An XCS also executes a niche GA which only operates on the matching pool rather than on the entire population. Wilson developed an XCS by making some changes to an LCS based on the following motivations [53]:

- Since payoff levels of various environmental niches are not identical, a consequence of using payoff as fitness will cause high-payoff niches to dominate the population. However, in some cases, a low-payoff classifier is probably the best one for its environmental niche. In terms of accuracy of payoff, an XCS enables classifiers to learn without regard for the niche payoff level.
- Basing fitness on accuracy of payoff allows an XCS to construct a more complete map of payoff environment, then benefit action selection since more alternative predictions are produced.
- Using accuracy can help a system to distinguish between accurate classifiers and over-general classifiers with the same strengths, whereas a traditional LCS is unable to make such a distinction.
- The GA in an XCS acts upon match sets, and thus results in an approximately equal allocation of classifiers to the various niches.

The execution cycle of an XCS, like that of an LCS, consists of three major subsystems: *the performance* (rule and message) subsystem, *the reinforcement* (credit assignment) subsystem, and *the rule discovery* subsystem. Fig. 4.7, adopted from [113], illustrates an

XCS.

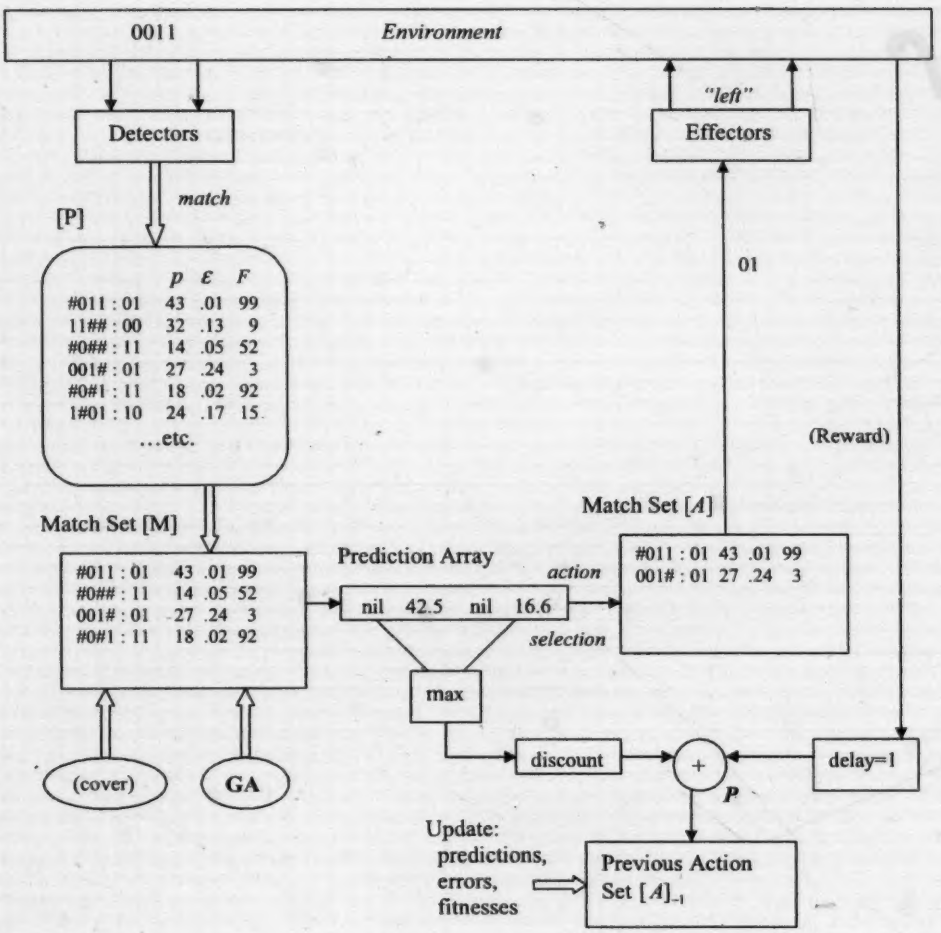

Fig. 4.7. A schematic illustration of XCS reproduced from [113]

4.4.1 The Performance Subsystem

The performance subsystem in an XCS is similar to the rule and message subsystem of an LCS. The operation cycle can be described as follows:

1. Detectors collect an environmental message and encode it as a binary string.
2. A match set [M] is formed from the classifiers in the population [P] whose conditions match the input string.
3. The action set [A] of classifiers in [M] is formed based on the prediction $P(\varphi_i)$ for each action φ_i represented in [M].
4. The selected action is translated into the output of the system through effectors.

If no action is selected, i.e., [M] is empty, a classifier is randomly generated through so called covering.

4.4.2 The Reinforcement Subsystem

In an XCS's reinforcement subsystem, the bucket brigade algorithm is replaced by a Q-learning-like system to update the p, ε, and F in the previous time step's action set $[A]_{-1}$. As shown in Fig. 4.7, the quantity P is updated by:

$$P = R(t-1) + \gamma \max(P(\varphi_i)), \tag{4.15}$$

where R is the reward, and γ is the discount factor ($0 < \gamma \leq 1$).

For each classifier in $[A]_{-1}$, the prediction p_j is updated as follows:

$$p_j = p_j + \beta(P - p_j), \tag{4.16}$$

where β is a learning rate parameter ($0 < \beta \leq 1$).

The prediction error ε_j is updated as follows

$$\varepsilon_j = \varepsilon_j + \beta(|P - p_j| - \varepsilon_j) \tag{4.17}$$

The fitness F_j is updated in three steps:

Step 1:

$$k_j = \begin{cases} e^{\frac{(\varepsilon_j - \varepsilon_0)\ln\alpha}{\varepsilon_0}} & \text{if } \varepsilon_j > \varepsilon_0 \\ 1 & \text{otherwise} \end{cases} \tag{4.18}$$

in which k_j is the accuracy of each classifier, and ε_0 is the criterion up to which the classifiers are considered to be accurate. For error above ε_0, k_j declines exponentially. The rate is set such that k_j at $\varepsilon_j = 2\varepsilon_0$ equals α, $0 < \alpha < 1$.

Step 2:

$$k_j' = \frac{k_j}{\sum k_{[A]}}, \tag{4.19}$$

where k_j' is the relative accuracy.

Step 3:

$$F_j = F_j + \beta(k_j' - F_j) \tag{4.20}$$

where β is the learning rate.

4.4.3 The Rule Discovery Subsystem

In an XCS, a niche GA is used to act on the match set [M], as shown in Fig. 4.7. Genetic operators are performed on two classifiers selected from [M] with probabilities proportional to their fitness.

The advantage of the niche GA is described in [113]: "... the rate of incidence of the GA is controlled with the aim of allocating classifier resources approximately equally to the different match sets (such an allocation being consistent with the purpose of forming a

relatively complete mapping). This cannot in general be achieved if the GA simply occurs with a certain probability in each match set. Depending on the environment, some match sets (niches) may occur much more often than others. Instead, the GA is run in a match set if the number of time-steps since the last GA in that match set exceeds a threshold. As a result, the rate of reproduction per match set per unit time is approximately constant–except in the most rarely occurring match sets."

Chapter 5

LCS-based Robot Controller

Robot navigation is usually complicated by one or more of the following [6]:

- information about the environment is available as a stream of potentially noisy and extraneous data;
- instantiation of action needs to be continuous, usually real-time;
- goals are inexactly or implicitly defined;
- payoff is sparse in that long sequences of action occur between reinforcement.

In order to handle the above problems, the robot controller should be able to:

- use the environmental information encountered while the robot is travelling to refine the environment model;
- provide a distinct action corresponding to present inputs in each process iteration;
- discover the goal-relevant regularities relative to the environment;
- assign appropriate credit to stage-setting situations that occurr during the travel.

Solutions that assume a priori knowledge of the environment are not sufficiently flexible to deal with situations in which an unforeseen obstacle blocks a path to the goal. It is better if the robot itself learns from the environment. Considering the above descriptions, an LCS is a good choice for an autonomous robot controller. It can effectively interact with the environment and eventually endow the robot with learning ability to carry out the navigation task.

In this chapter a conventional LCS used for obstacle avoidance is described. Navigation with obstacle avoidance behaviours using wheeled robots were studied in early ER work and continue to be the benchmark test for novel robot controllers. The objective of this study is to move a robot in its environment, avoiding obstacles. To reach this objective an LCS-based controller is developed and a learning stage is applied.

Fig. 5.1. Pioneer 2-DX robot

The robotic platform used in the experiment is a Pioneer 2-DX simulated in the SRI robotic simulator. Pioneer 2-DX (as shown in Fig. 5.1) is a mobile robot, containing all of the basic components for sensing and navigation in a real world environment: battery power, motors, wheels, postion/speed encoders, integrated sensors, and accessories. The robot is controlled either through an onboard microcontroller or by server software. Sixteen sonar sensors are distributed around the body of the robot, 8 on the front side and 8 on the back. The sensors detect the presence of objects by emitting ultrasound waves and

measuring a reflected signal.

Fig. 5.2. The layout of a SRIsimulator

A simulator is used to test the feasibility of the learning system by which a simulated robot is controlled. The SRI simulator (shown in Fig. 5.2) provides a virtual replacement for a real Pioneer 2-DX robot when it is not practical to achieve certain learning assignments. The SRI simulator acts much like a real robot, not only in its relative size and sensing of sonars, but also in realistically simulating wheel slippage and other real sensor errors. The simulator offers a realistic platform which includes the pitfalls of real-world mobile robotics. In this study, a simulated robot is controlled by the motor commands provided by a LCS. Two types of simulated environments, a pilaster (shown in Fig. 5.3) and a polygon (shown in Fig. 5.4) environments, are adopted to test the robot's performance. The pilaster environment is drawn in the same form as in [70, 71] to allow comparison. No comparison of learning time T is provided in this study since the simulators and computing platforms differ. The major concern is whether the robot is learning or not, and the robot

trajectories are used to make this assessment.

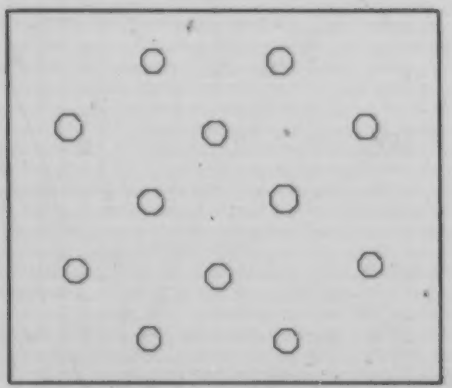

Fig. 5.3. The layout of a pilaster environment

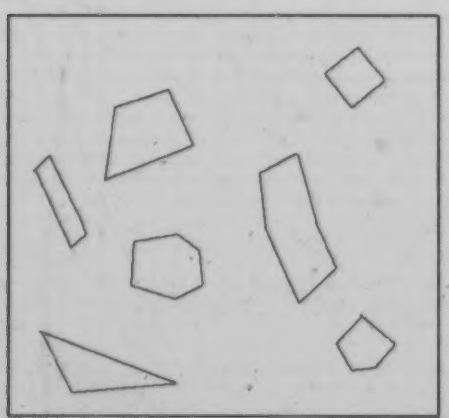

Fig. 5.4. The layout of a polygon environment

5.1 Representation: Encoding and Decoding

An important problem in mobile robotics is the depiction of environmental regularities by a practical representation. An LCS, like all learning systems, needs to represent real world knowledge in a form that can be manipulated. To solve the robot navigation problem, the representation of environmental information and corresponding movements as binary strings of 1s and 0s is considered optimal since sensory inputs and motor commands can be processed or implemented at a microprocessor level.

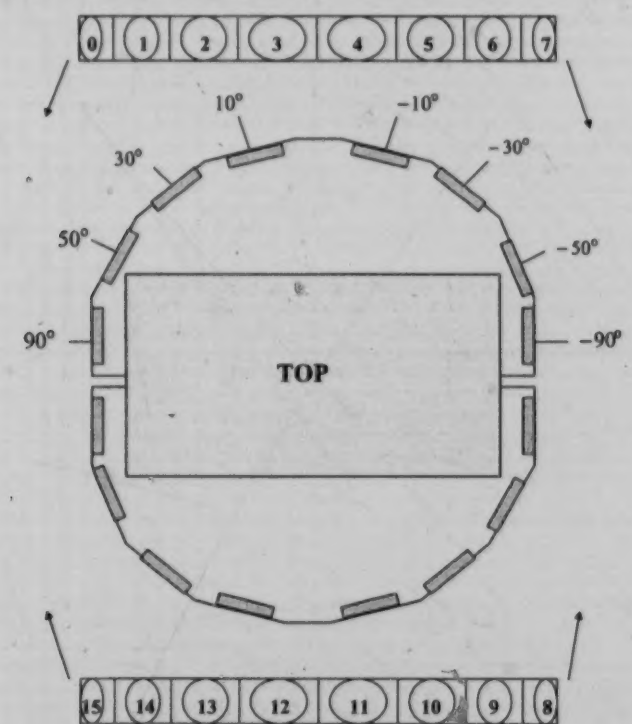

Fig. 5.5. The placements of sonar

Sixteen proximity range sonars are placed around the body of the robot (as shown in Fig. 5.5). The 8 front sonars are spaced evenly from 90 degrees at the left side around robot's front to -90 degrees at the right side, and 8 back sonars are distributed from -90

degrees at right side to 90 degrees at left side. The maximum range of the proximity sonar is 3000 *mm*. At each iteration, the robot maintains its motor command by continuing its current motion for 1000 *ms*. The signals collected by sonars ($s_0 \sim s_{15}$) at each iteration are transformed into binary values by a thresholding process. The threshold Θ is set as 1000 *mm*: a sonar range with a value greater than Θ is encoded as 0, otherwise 1. Typically if Θ is too large, sensing might receive uniform representations in a congested area and therefore discover fewer environment niches. With $\Theta = 1000$ *mm*, the robot still has enough time to provide reflex behaviours since the maximum possible velocity of the Pioneer 2-DX is 300 mm/s. An example of an encoded environmental message can be illustrated as follows.

s_0	s_1	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}
1	0	1	1	1	0	0	0	0	1	1	1	0	1	1

To translate the action of a classifier rule into usable motor commands, a novel decoding mechanism is proposed [74]. The robot's actions are determined by three effectors (represented by 6 bits): one to control the turning angle (4 bits), another to control the speed (1 bit) varying between 100 mm/s and 200 mm/s, and the third to control the forward/backward direction of travel (1 bit). The turning angle is expressed in increments of 5° using a 4-bit binary code as shown in Table 5.1. In order to prevent the robot from unnecessary turning, the proposed symmetrical encoding scheme has more entries for actions with smaller turns, increasing the probability that the robot will move ahead.

The total length of each rule is 22 bits plus a real value of strength. The rule population is initialized using a uniform random distribution by filling all positions (conditional parts with 0, 1, and #; action parts with 0 and 1). A population size of N=800 is used, and the initial strengths of all rules are set to a common value $S_i(0)$=100. An example of a complete classifier rule is shown below:

Turning angle	Codes	Probability
Left 15°	1100	0.0625
Left 10°	1000, 1110	0.125
Left 5°	0100, 1010, 1101	0.1875
Straight 0°	0000, 0110, 1001, 1111	0.25
Right 5°	0010, 0101,1011	0.1875
Right 10°	0001, 0111	0.125
Right 15°	0011	0.0625

TABLE 5.1

Encoding of the robot turning angle

10#1000#11#0##10	1011	0	1	99
Conditional part λ to match the sonar inputs	Turning Angle (Right 5°)	Speed (Slow)	Direction (Forward)	Strength

5.2 Matching Process

After an encoded environmental message X is generated, the matching process starts.

- All classifiers in the population try to match its condition λ with the binary environmental string. The matching is executed bit by bit; if the entries at all non-# positions are identical, the message satisfies the condition. The bit match function is defined as

$$f_m(x,y) = \begin{cases} true, & \text{if } x=y \\ \\ false, & \text{otherwise} \end{cases} \tag{5.1}$$

The j-th rule match function is

$$F_m(X, C_j^{\lambda}) = \prod_{i=0}^{15} f_m(x_i, \lambda_i), \tag{5.2}$$

where x_i is the i-th bit of X, and λ_i is the i-th bit of condition in classifier C_j. F_m is applied to each classifier in the population; those with non-*false* function values are put aside to compose the matching pool [M].

- To handle situations where no rule matches the input, the *creation* process [114] is used. The conditional part of a newly created classifier rule is a copy of the current input string, except that '#' symbols are inserted with a probability corresponding to the current percentage of all "don't cares" in the population. The action of the new rule is set randomly, and its strength is set to the mean strength of the current population. To make room for the newly created rule, one rule is deleted from the population according to the probability distribution over the inverse of the strengths of all rules. The probability of deletion is given by

$$P_{del,i} = \frac{\frac{1}{S_i}}{\sum_{j=1}^{N} \frac{1}{S_j}} \tag{5.3}$$

5.3 Credit Assignment Process

The modifications of strength $S_i(t)$ are performed by a process described below:

- The winning rule is selected from the pool of matches using the bids calculated by Eq. 4.9 with the risk factor ρ set to 0.5. The probability of selection of a rule, P_j, is equal to its bid divided by the sum of bids of all rules in the matching pool [M].

$$P_j = \frac{B_j}{\sum B_{[M]}} \tag{5.4}$$

- The reinforcement component adjusts rule strengths in accordance with the payoff P_f received from the environment after performing an action. A reward R_i is distributed to the winning rule, whose strength is reduced by the amount of its bid, B_i,

as in Eq. 4.13. R_i could be either positive or negative to execute reinforcement or punishment, respectively.

- All other rules are updated by Eq. 4.14 with Tax_{bid} and Tax_{life} both set to 0.5.

5.4 Rule Discovery Process

The GA can be invoked after a number of collisions has passed [70]. In this experiment the GA is invoked after 10 collisions and works as follows:

- A classifier rule C_i in the population is selected using a probability distribution over the strengths of all rules in the population. The selection probability P_i is calculated as follows:

$$P_i = \frac{S_i}{\sum_{j=1}^{N} S_j} \tag{5.5}$$

  where N is the population size.
- With a probability of crossover, P_c, the classifier rule C_i is submitted to a one-point crossover with each rule in the population. Each parent's strength is reduced by one-third and the offspring's initial strength is set to the sum of the reductions.
- With a probability of mutation, P_m, the mutation operation is applied to each individual. The strength of offspring resulting from mutation is not changed.
- When an offspring is added to the population, another rule is deleted using a probability distribution over the inverse of the strengths of all rules to keep the size of the population constant. The deletion probability P_i is given as:

$$P_{del,i} = \frac{\frac{1}{S_i}}{\sum_{j=1}^{N} \frac{1}{S_j}} \tag{5.6}$$

In summary, the GA chooses higher-strength classifier rules to reproduce, and generates

new individuals by their recombination and mutation. In the simulations, P_c is set to 0.7 and P_m to 0.025. The value of P_m is slightly lower than in typical GAs to limit disruption of good rules in the population.

5.5 Implementation

The simulated robot and environment are used to verify the algorithm. Fig. 5.6 shows a flowchart of the implementation.

Step 1: The initial classifier population is created by a random process so that the robot controller does not have prior knowledge of the environments being navigated. Each classifier in the population consists of binary strings of 22 bits, as well as a real strength value. In this step a "$Counter_{collision}$" is defined which records the number of collisions the robot has encountered, beginning at zero.

Step 2: Sonar readings are encoded as 16-bit binary strings by a thresholding process.

Step 3: A matching process is applied to each classifier in the population in terms of Eq. 5.2. If no rules in the population match the input, the creation process is engaged.

Step 4: A competition among the rules in the matching pool is performed. The winner determines an action; its action part is decoded to motor commands, i.e., turning angle, velocity, and direction.

Step 5: The strengths of all the rules except the winning classifier are updated in terms of Eq. 4.14.

Step 6: The status of the robot is checked (e.g., whether the left or right motor is stalled). If the left-motor or right-motor is stalled, R_i is set to a fixed negative value and the $Counter_{collision}$ is incremented by 1. If neither motor is stalled, R_i is set to a fixed positive value. After the rewards value is determined, the strength of the winning rule is updated by Eq. 4.13.

Step 7: The value of $Counter_{collision}$ is checked. If the $Counter_{collision} \geq 10$, then a GA is triggered, and the $Counter_{collision}$ is reset to 0.

Step 8: The execution cycle proceeds from step 2.

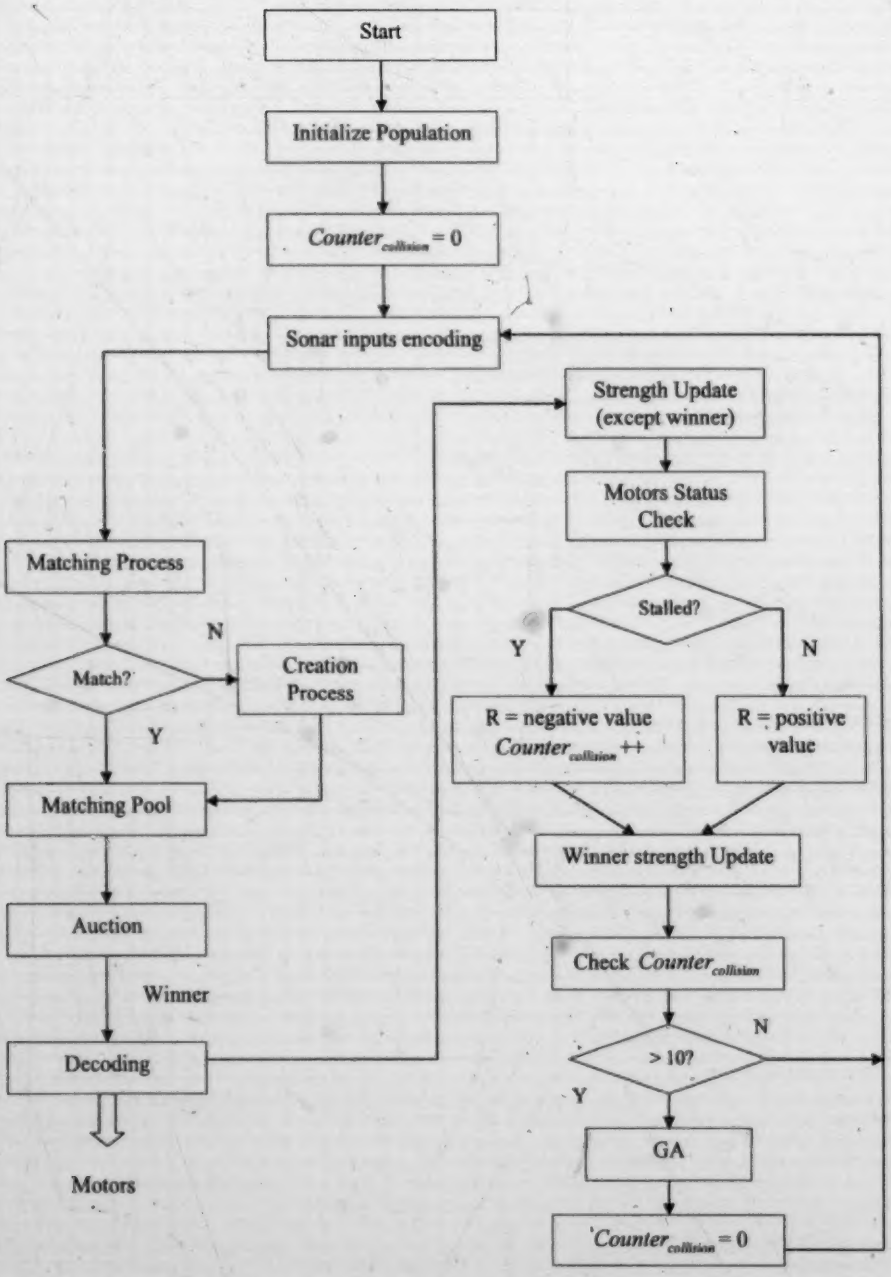

Fig. 5.6. Implementation cycle of obstacle avoidance by a LCS controller

5.6 Simulation Results

The robot starts to move from a corner of the arena. Fig. 5.7 and Fig. 5.8 show the trajectories at the beginning of the learning stage in both pilaster and polygon environments. Many collisions occur during learning at which time the classifier system receives high penalties that are used to modify and improve its performance.

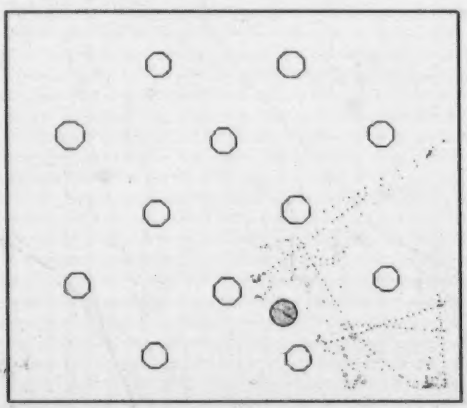

Fig. 5.7. The trajectory of robot at the beginning of learning (pilaster)

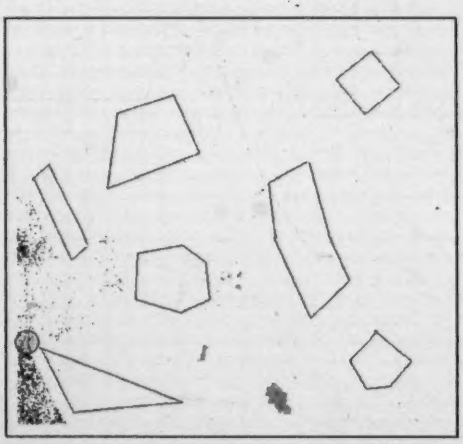

Fig. 5.8. The trajectory of robot at the beginning of learning (polygon)

Since the objective of the study is to provide a robot with learning ability, the behaviour of a robot can be characterized by the trajectory describing its movement. After the learning stage, the robot finds equilibrium in an almost circular orbit around the pilaster as shown in Fig. 5.9 and Fig. 5.10, similar to the results reported in [70]. Fig. 5.11 and Fig. 5.12 show the robot's path in the polygon environment.

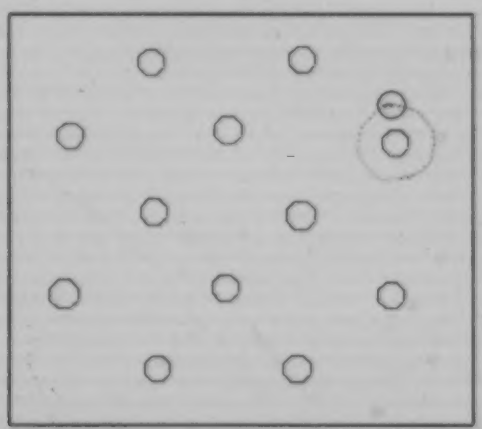

Fig. 5.9. The trajectory of the robot in an equilibrium state (pilaster)

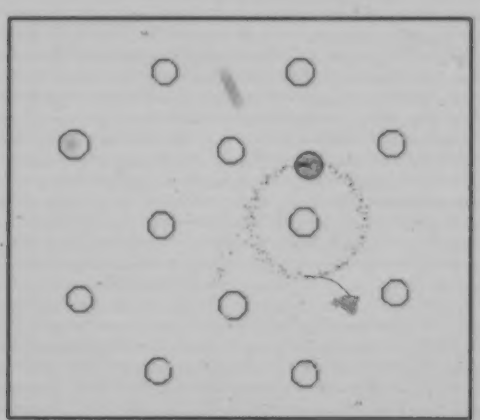

Fig. 5.10. The trajectory of the robot in another equilibrium state (pilaster)

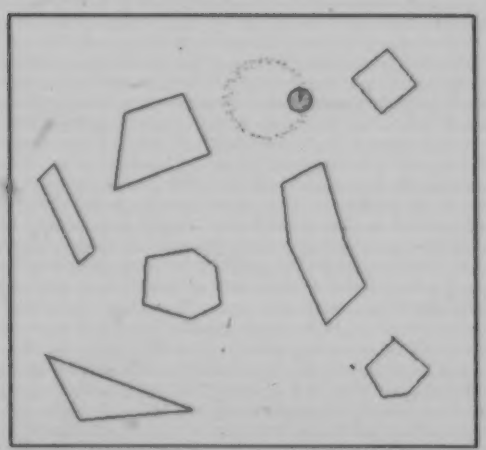

Fig. 5.11. The trajectory of the robot in an equilibrium state (polygon)

Fig. 5.12. The trajectory of the robot in another equilibrium state (polygon)

Fig. 5.13 is a graph showing the cumulative number of iterations without collision during the learning phase (in the pilaster environment). After the robot stops learning (around iteration $t = 60000$), the number increases exponentially with time. Fig. 5.13 also shows the initial stage before the robot converges to the the circular orbit: the time between collisions is increasing, which indicates that the robot is learning. After this period, no more collisions and hence no more learning occurs. In [70], a similar measurement was adopted. The non-collision-iterations (NCM) showed an overall tendency to increase with the time, however, the authors plotted only part of learning process. Since the simulated environments and computer platforms differ, it is not practical to compare the NCM from different studies. However, the NCM should increase if the robot is learning.

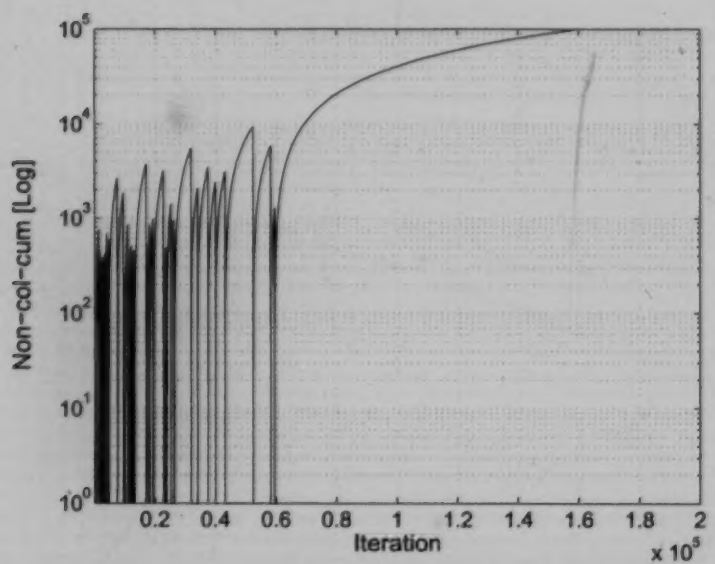

Fig. 5.13. Non-collision-cumulation with large horizontal scale (pilaster)

Chapter 6

Enhanced LCS Controller with Perceptual Memory

6.1 Shortcomings of the LCS Controller

Based on the results described in Chapter 5, it is apparent that the conventional LCS provides the robot with the ability to learn. However, there are problems that significantly affect its performance, especially in more complex environments. The most significant deficiencies are *local minima*, *rule destruction* and *generalization* [59].

6.1.1 Local Minima

Robot actions quickly converge to the same circular orbit. The problem arises because the robot ceases to learn new rules and dwells in a local minimum. It is useful to analyze the behaviours of the LCS after convergence. The strength of each LCS in the equilibrium state is denoted as S_{es}. Then the strength of winning rule is

$$S_{es,win}(t+1) = S_{es,win}(t) + R_i(t) - B_{es,win}(t) \tag{6.1}$$

with the bid of the winning rule, $B_{es,win}$, calculated by Eq. 4.9. The value of the risk

factor ρ is typically such that $B_{ex,win}(t)$ is slightly smaller than $R_i(t)$. Therefore, some rules dominate the rule base as the winning strengths keep growing. The rules other than winners lose the competition every iteration by having their strengths levied by either Tax_{life} or Tax_{bid}. This way, the population develops monotonous rules that recognize only a few input messages.

6.1.2 Rule Destruction

As the number of environmental messages increases, so does the complexity of the learning task. However, the population size cannot be increased without limits to address this increased complexity due to the associated increase in execution time. The resulting effect is that some useful rules are replaced with new ones and therefore the system cannot distinguish between a sufficiently large number of classes.

During the learning process, the goal is to preserve the learned rules and learn new rules. The learned rules include the classifiers whose conditions match some specific environment niches and those actions that were rewarded. A GA is used to discover new rules, but it may destroy rules that have been already learned in the process of maintaining diversity in the population. The robot therefore may not execute the correct actions when it approaches the same niches, having "forgotten" due to rule destruction.

6.1.3 Generalization

The creation and rule-discovery processes tend to generate some general rules that are undesirable when attempting to discriminate between many sonar inputs. If the conditional portion of a rule is too general, it may provide an action inappropriate to the input, even though the few fixed conditions are met.

In the creation process, # symbols are inserted randomly into the conditional part of a newly created classifier. In addition, the mutation operator of the GA changes the ternary code in the following way

'0' → '1' or '#'

'1' → '0' or '#'

'#' → '0' or '1'

Therefore, some # symbols could be obtained in the process of mutation. The rules generated by these two processes could match many environmental inputs. However these rules provide appropriate actions to certain matched environmental messages, but inappropriate actions to others. The robot is therefore not able to distinguish the similar niches in the environment.

6.2 Enhanced LCSs

6.2.1 Enhancements

To address the problems mentioned above, the following enhancements are proposed and implemented.

Escaping from Local Minima

As mentioned earlier, the criteria used to invoke the GA is the number of collisions. However, once the system converges to equilibrium (e.g. a circular orbit), the rule discovery process is not activated since no further collisions occur. To allow escape from local minima, it is proposed that the GA be invoked after a certain number of iterations rather than only upon a specified number of robot collisions.

To decrease the probability of disruption of good rules, the number of iterations between applications is increased as the system learns. Because most rules at initialization are inappropriate, applying the GA helps the system find the more appropriate rules. However, as time progresses, most rules in the rule base are correct and application of the GA need not (and should not) be too frequent.

Rule Protection

Inspired by the work of McAulay and Oh [63], "No-replace" and "No-parent" flags are applied to index classifier rules that have high strengths and provide appropriate actions. These rules are not replaced in later generations and do not become parents in the next generation, preventing useful rules from being replaced and groups of similar rules from dominating the rule base, respectively.

As the robot travels, most of the robot's actions play a stage-setting role for goal attainment. By applying the *bucket brigade* algorithm, such actions receive a payoff but are offset by a paying bid with a small positive balance. In order to protect useful rules, it is proposed to place variable positive reward values on winning actions. Among the actions following immediately after a collision, the classifier whose action first leads to the robot moving away from the obstacle receives a much higher reward than others, and thus is indexed with "No-replace" and "No-parent" flags. A similar situation holds for an animal. When an animal hits an object and feels pain, the first instinctive movement that drives it away from harm gives the animal a distinct impression. Later, the animal will execute the same movement when it senses a similar danger, i.e. it will remember the corresponding rule.

Specialization

To handle rules that are too general, a special GA is applied to those rules that provide incorrect actions leading to collisions. The genetic operators are only applied to the action part, while the condition part is replaced by the current sonar inputs. This way, a new specialized rule is created with exact environmental inputs and a new action.

6.2.2 Simulations

The proposed enhancements have been implemented, as shown in Fig. 6.1.

Step 1: The initial classifier population is created by means of a random process so that

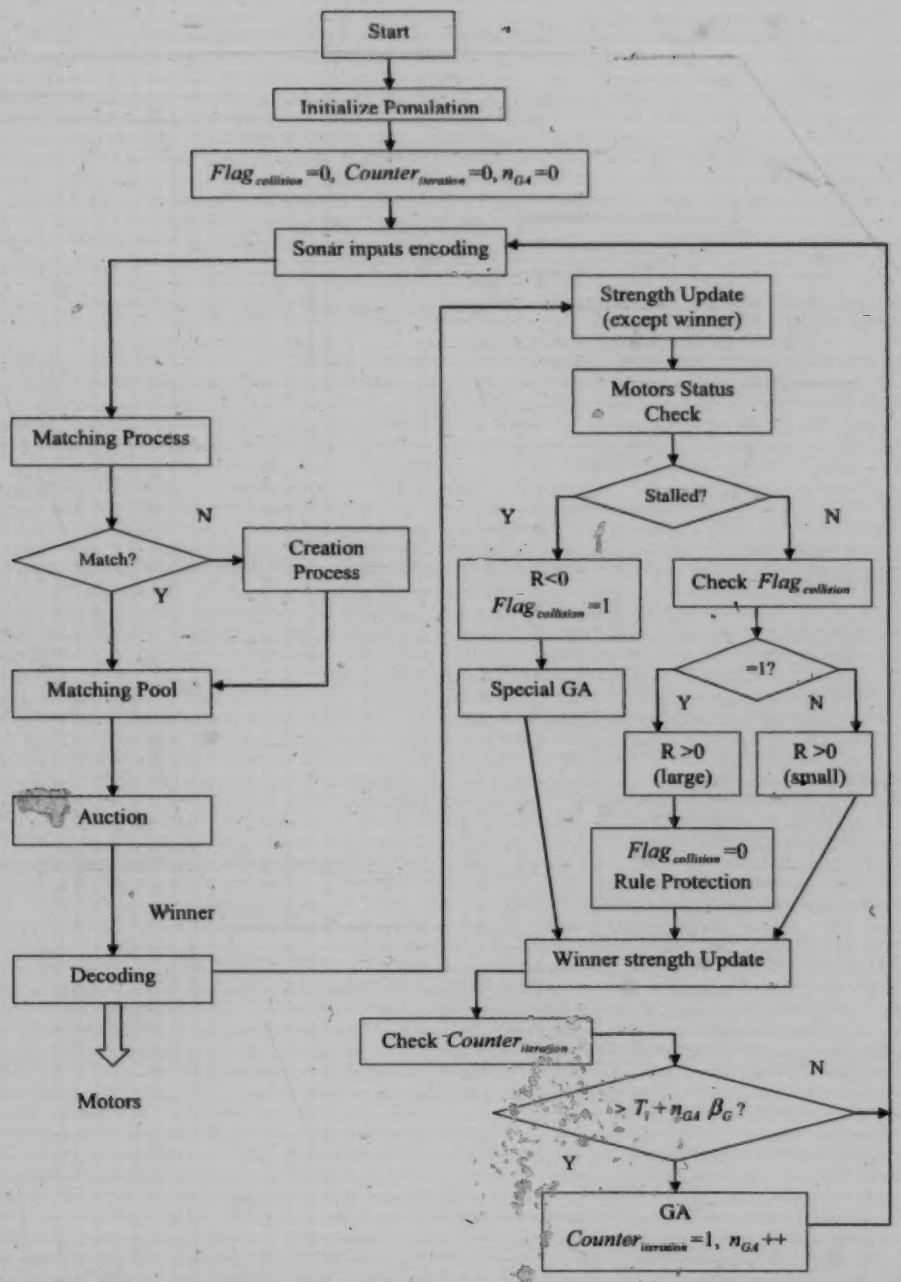

Fig. 6.1. Implementation cycle of obstacle avoidance by the enhanced LCS controller

the robot controller does not have prior knowledge of the environments being navigated. Each classifier in the population consists of 22 binary bits as well as a real strength value. In this step, a counter of collision, $Counter_{collision}$, and a counter of GA execution, n_{GA}, are defined and set to zero. T_i is pre-defined as the initial number of iterations after which a GA is triggered. Another variable, $Flag_{collision}$, is initialized as 0. $Flag_{collision}$ describes the robot status in the immediate past state: if collision occurred, $Flag_{collision}$=1, otherwise $Flag_{collision}$=0.

Step 2: Sonar readings are received and encoded as 16-bit binary strings by a thresholding process.

Step 3: A matching process is applied to each classifier in the population in terms of Eq. 5.2. If no rules in the population match the input, the creation process is engaged.

Step 4: A competition among the rules in the matching pool is performed. The winner determines an action; its action part is decoded to motor commands, i.e. turning angle, velocity and direction.

Step 5: The strengths of all the rules except the winning classifier are updated in terms of Eq. 4.14.

Step 6: The status of the robot is checked in this step (e.g., check for stalled motor). If the left-motor or right-motor is stalled, the strength of winning rule is updated by means of Eq. 4.13, in which R_i holds a fixed negative value. A special GA is applied to the winning classifier. $Flag_{collision}$ is set to 1 because a collision just occurred. If neither motor is stalled, the value of $Flag_{collision}$ is checked:

- R_i holds a fixed large positive value when $Flag_{collision} = 1$, the rule protection process starts and $Flag_{collision}$ is reset to 0 because no collision occurs in the current iteration.
- R_i holds a fixed small positive value when $Flag_{collision} = 0$.

After the rewards value R_i is determined, the strength of the winning rule is updated by means of Eq. 4.13.

Step 7: The value of $Counter_{iteration}$ is checked. If $Counter_{iteration} \geq T_i + n_{GA}\beta_G$, where $\beta_G \geq 1$, the GA is triggered. $Counter_{iteration}$ is reset to 0, and n_{GA} is incremented by 1.

Step 8: The execution cycle proceeds from step 2.

The enhanced LCS using the proposed modifications provides satisfactory results. During the learning stage, the robot learns while exploring the entire environment, as shown in Fig. 6.2 and Fig. 6.3. In order to prevent the robot from converging to a circular orbit, the GA must be constantly applied because the GA is the only method to bring variation into the action part of each rule. Although the strength of some rules may keep growing in some niches, application of the GA leads the new actions of those rules. However the new actions may cause collisions in same or other niches; the rules then receive penalties from these occurrences. The trend of dominating the rule base is therefore obstructed.

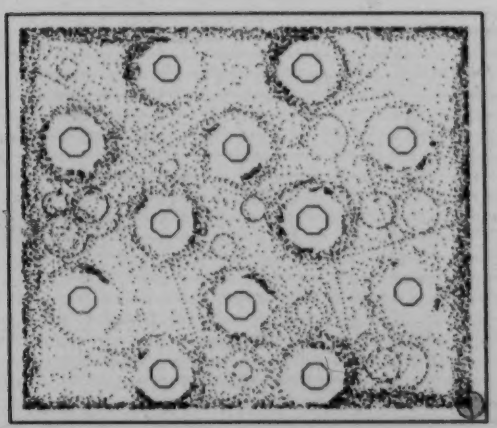

Fig. 6.2. Robot trajectory using enhanced LCS during the learning period (pilaster)

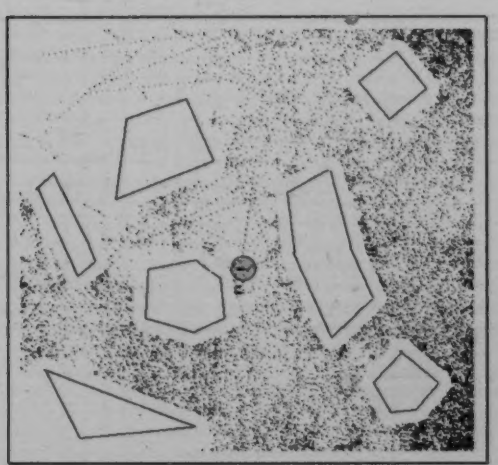

Fig. 6.3. Robot trajectory using enhanced LCS during the learning period (polygon)

Fig. 6.4 and Fig. 6.5 show the robot's paths in pilaster and polygon environments, respectively, after the learning stage: no collisions occur while the learning process continues to explore the environment.

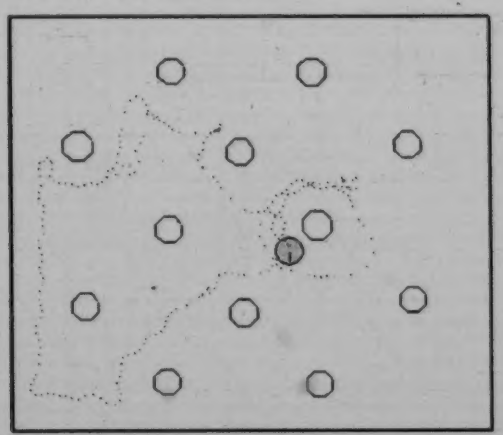

Fig. 6.4. Robot trajectory using enhanced LCS after the learning period (pilaster)

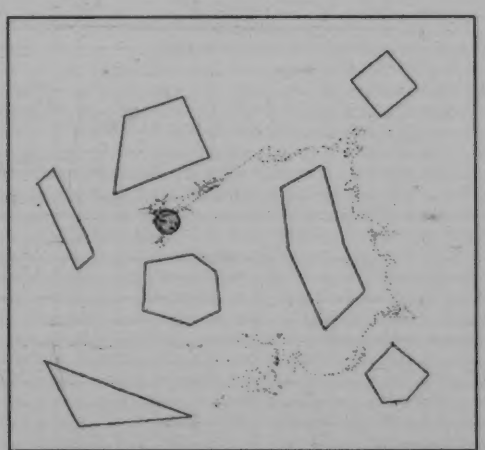

Fig. 6.5. Robot trajectory using enhanced LCS after the learning period (polygon)

Fig. 6.6 and Fig. 6.7 show the cumulative number of iterations without collision (NCM) during the learning of the enhanced learning classifier system (ECS) in different environments, pilaster and polygon.

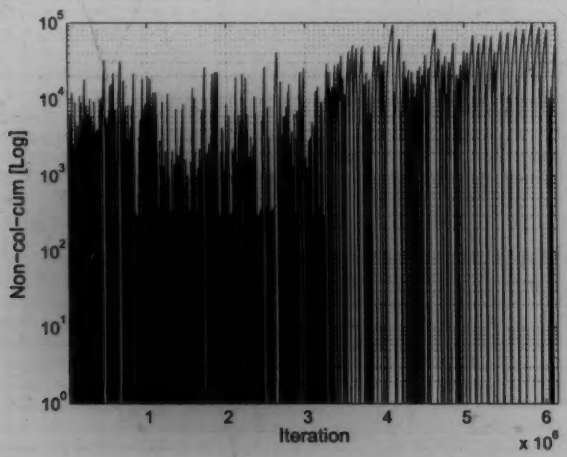

Fig. 6.6. Non-collision-cumulation (pilaster)

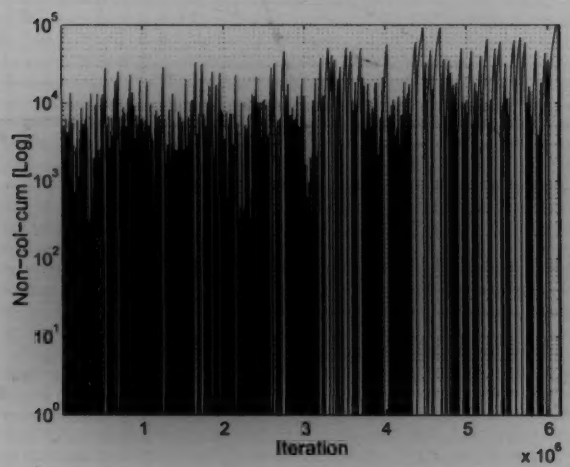

Fig. 6.7. Non-collision-cumulation (polygon)

After iteration 4000000, the curve in Fig. 6.7 is more oscillatory than that of Fig. 6.6. This implies that the robot suffers more collisions in the polygon environment. This may be because the polygon is an unsymmetrical environment; in some congested areas, e.g., the place below the triangle in the left-bottom corner, the robot encounters many collisions and receives a high penalty.

Once a collision occurs, several other collisions follow closely while the specialization enhancement creates a new action with an updated conditional. If the action of the current winning rule is appropriate, the robot moves away from the obstacle. Otherwise, the robot will approach the obstacle again, and the specialization process restarts.

6.2.3 Analysis

In contrast to XCS, which utilizes the population to form a complete mapping of the payoff (rewards) environment by RL, the proposed enhanced LCS extends the Michigan-style LCS which maximizes classifier payoff [22]. Although several enhancements are proposed to enrich the performance of the Michigan-style LCS, the enhanced LCS still uses the same formulation and the primary working principle is still to maximize the reward when the robot is interacting with the environment. A Michigan-style LCS employs strength as a predictor of payoff in the auction process and as a fitness function in the GA. In XCS, the strength of payoff prediction is replaced by three parameters: payoff prediction, prediction error, and fitness.

The payoff levels for various niches in the same environment may be different. If the fitness is purely based on classifier strength, classifiers in the higher-payoff niches may dominate the population. Based on the fitness of classifier accuracy, XCS evolves classifiers without considering the niche payoff level. In the proposed enhanced LCS, a similar role is played by the rule protection enhancement: it keeps good classifiers in lower-payoff niches to prevent classifiers in higher-payoff niches from taking over the entire rule base.

Another advantage of the accuracy based fitness of XCS is that it avoids giving inappropriate priority to classifiers that are too general since they could be matched more often.

In fact, the specialization process of the proposed enhanced LCS decreases the generality of the population during learning: when applying a GA to produce new general classifiers, specialization influences the evolution.

To summarize, the proposed enhanced LCS achieved performace improvements similar to those of XCS. However, the proposed system does this in an implicit way by introducing counters and flags rather than explicitly changing the way fitness is calculated and a GA is applied. As a result, the enhanced LCS possesses qualitatively similar characteristics with less effort compared to XCS.

6.3 An LCS with Perceptual Dynamics

To speed up the learning process and to deal with oscillations occurring after each collision, it is proposed to include robot perceptual dynamics directly as inputs into the LCS.

In a conventional LCS, the response of the system at each moment is uniquely determined by the current state of the environment. However, this is undesirable in the case of robot navigation due to the dynamic nature of the robot-environment relation. A dynamic robot control system should maintain past state information, so the response of the robot exposed to identical environmental conditions can vary at different times, depending on the history of its movements.

An example of a dynamic control system is a dynamic neural network (DNN). Since the first use of a DNN to evolve adaptive robot behaviour [5], a large amount of work has been done on evolving successful simulated robot control systems [48, 69]. Because these networks have feedback connections, they are sequential rather than combinational. In other words, they exhibit temporal behaviour.

Inspired by the DNN, it is proposed to add perceptual dynamics to the LCS by including a delayed input as a part of the rule conditionals of the rules. Time delay is an important factor in dynamic systems [88] and plays a crucial role in robotic and cognitive systems [106]. In the case of robot learning, it contributes to stability of the learning process and improves response to the changing environment.

The DNNs has a network structure that accommodates feedback connections. In contrast the rule and message constituents are fundamental to the structure of an LCS. Accordingly, past sonar inputs can be included in the conditional part of each classifier. Due to the performance restrictions, only the sonar signals of the immediate preceding reading are considered. Thus the environmental message includes two portions of sonar inputs: past state and current state. In each iteration, current sonar signals are stored and applied as past state inputs in the next iteration. The rule population is randomly initialized. An example of a dynamical classifier rule is shown below. The rest of the LCS structure remains unchanged.

10#1000#11#0##10	10#1011#11#0##10	1011	0	1	156
λ of current inputs	λ of past inputs	Right 5°	Slow	Fwd	$S(t)$

Experimental results confirm that the proposed dynamic learning classifier system (DCS) explicitly alleviates oscillations occurring after each collision. To compare the LCS with and without time delay, the pilaster environment is employed. Moving averages of the frequency of collisions are plotted in Figs. 6.8 and 6.9. Both collision occurrence frequencies decrease as the learning progresses. However, the DCS suppresses oscillations right from the beginning of the learning process (Fig. 6.9).

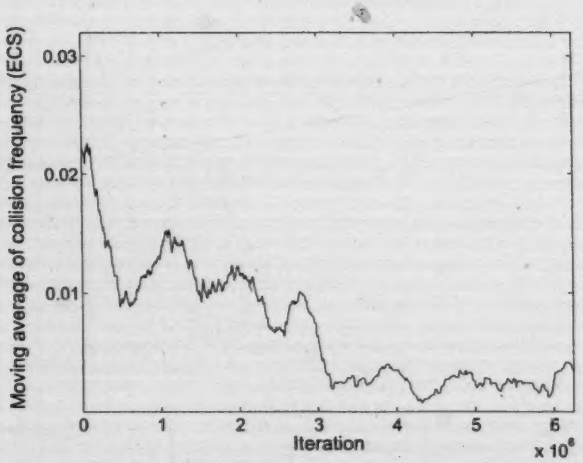

Fig. 6.8. Moving average of collision occurrence frequency by enhanced LCS (pilaster)

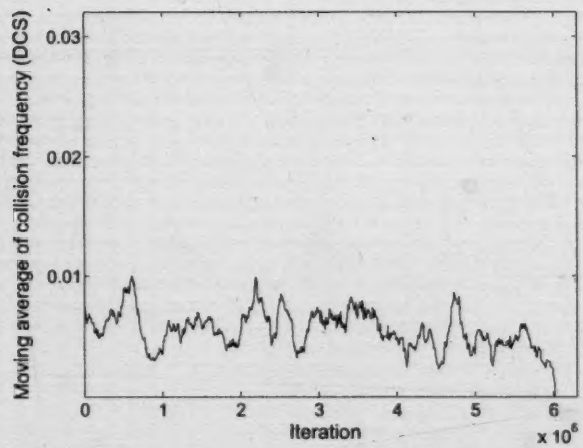

Fig. 6.9. Moving average of collision occurrence frequency by enhanced DCS (pilaster)

As mentioned above, in the static (enhanced) LCS a number of collisions follow closely after a collision. However the inertial nature of the DCS prevents the robot from taking identical action (likely leading to another collision) after a collision has occurred. In such circumstances, the system initiates a new action different from the current (inappropriate) action and, consequently, other collisions do not follow immediately.

To test this hypothesis, all the consecutive collision sequences (CCSs) during learning are collected. The length of each sequence represents the number of consecutive collisions. For instance, a total of 46857 collisions are recorded in the enhanced LCS learning process. There are 2 sequences with length 1, 8 sequences with length 2, ... , 7641 sequences with length 6, etc. Figs. 6.10 and 6.11 show the probability distributions of the consecutively occurring collision sequences by enhanced LCS and DCS respectively.

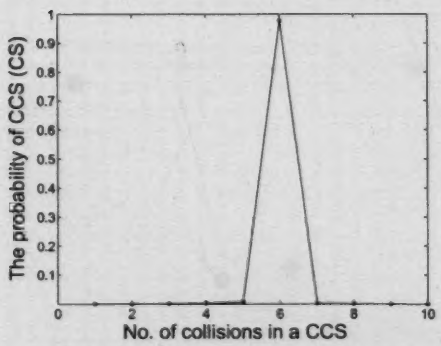

Fig. 6.10. The probability distribution of CCSs by enhanced LCS (pilaster).

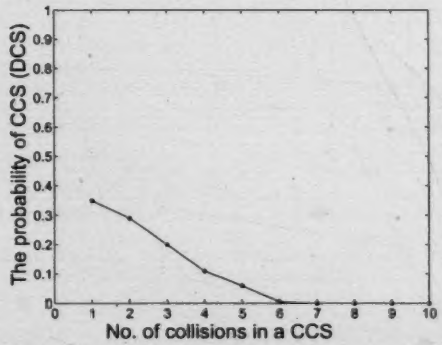

Fig. 6.11. The probability distribution of CCSs by enhanced DCS (pilaster)

In the case of the enhanced LCS, most collisions occur in sequences of length 6. On the other hand, most collisions in DCS learning happen in sequences with lengths shorter than 6. In other words, fewer collisions follow on the heels of a collision occurring in a DCS. It can therefore be concluded that inclusion of robot perceptual dynamics as inputs to the classifiers contributes to faster learning. To illustrate the DCS learning process, the

Iteration t	Conditional part λ	Action part φ	$S(t)$
1113109	1100#0##0##0#00#01#0#0#000000#0#	110011	571.345894
1113110	11000000000000001100000000000000	110011	72.623489
1113111	10000000000000011100000000000000	001111	177.083672
1113112	#00000#00########0#00#00#0#0###	001111	401.045685
1113113	#00#000000000#10#0000000##00000#	001111	70.325523
1113114	#00000#00########0#00#00#0#0###	001111	401.342896
1113115	0#000#00##00111###0#0###00##1###	001111	570.394684
1113116	0#####0###00011#0#00#0#00000#1##	001111	181.094307
1113117	#00000#00########0#00#00#0#0###	001111	101.637604
1113118	00#0##00#0#000####000#00###000#0	001111	245.578033
1113119	####000#0000##00#0##0000#00#0###	110011	873.430954
1113120	####000#0000##00#0##0000#00#0###	110011	873.730682
1113121	####000#0000##00#0##0000#00#0###	110011	874.030396

TABLE 6.1

Illustration of DCS learning process.

winning rules of the last few iterations before the equilibrium state (when no collisions occur) are recorded and shown Table 6.3.

Analysis:

The learning ability of a robot is closely related to its perception of the environment. Literally, there are two types of environments–Markov and non-Markov. In a particular environment, if the agent's immediate sensation provides all the information necessary to choose the best action in every situation, such environment is referred to as Markov [57]. In some other environments the agent's sensations are able to carry only partial information about the environment. As a result the agent cannot decide what is the best action with

respect to current sensory inputs. Such environments are said to be non-Markov. The simulated environments employed in this study are non-Markov because different niches of the environment may appear identical to the robot but may require different actions.

Learning from past experience is a desirable method for agents in non-Markov environments. In a Michigan-style LCS, the history of the system is typically expressed as an internal message list which behaves as a short-term memory. The messages are generated either from the environment or from the actions of other classifiers. As mentioned before, the LCSs in this study are S-R systems, so an internal message list of the actions from other classifiers can not be created. Therefore, how to provide the robot with temporal behaviour is a key point.

In [28] Foster developed explicit sequences of rules (meta-rules) to learn from experience. Temporal information is embedded into the rule base by meta-rules, rules that combine a number of rules in a particular sequence. "*Meta-rules enable an agent to make decisions based on sequences of rules rather than individual rules alone and endeavor to overcome the problem of hidden state ...*" [28]. However, extracting the meta-rules is not straightforward; they are extracted as an individual block within the entire complex control algorithm, by identifying repeating sequences of rules that were fired in the minimized history. This is necessary because the LCS part of the control system has been simplified by removing the rule discovery subsystem. Incorporating the evolutionary process of learning from experience through meta-rules is too complex to implement during the online robot navigation course, and therefore may not be practical.

Time can be a very important factor in the learning process. Many cognitive functions rely on time for appropriate execution. By adding time-delay as perceptual dynamics, the temporal nature of an LCS appears by way of certain dynamic properties. Specifically, the system has the ability to track the movements in the history; short-term memory is therefore built into the robot controller. Furthermore, incorporating time-delay is probably the simplest way to exhibit temporal behaviour.

Chapter 7

LCS-like NN Robot Controller

In some situations the propositional rules have been demonstrated to be inefficient in the implementation of intelligent systems [97]. Therefore, how to substitute the propositional rules while maintaining the advantages of LCS should be investigated. The fusion of NNs to compose the classifiers takes advantage of their characteristics of universal approximators and their ability to manipulate real numbers [70, 98, 109]. NNs present more than a mere substitution of other methods because their complex behaviours provide new possibilities to enrich the performance of LCSs [70]. In [70], each classifier is taken as an array that codifies the weights of two multiplayer perceptrons– an evaluation network which executes the matching, and an action network which generates the actual outputs to control the robot behaviours. This configuration becomes too complicated as the number of classifiers in the population grows, so the syntactic mapping of a Michigan-style LCS onto an NN [98] was adopted in this thesis to evolve obstacle-avoidance rules. The purpose of this work is to demonstrate the learning ability of an LCS-like NN and its applicability to robot obstacle avoidance.

7.1 An LCS-like NN

Inspired by Smith's study [98], an LCS can be represented as a competitive NN as shown in Fig. 7.1. This architecture suggests a simplified and functional LCS/NN analogy and

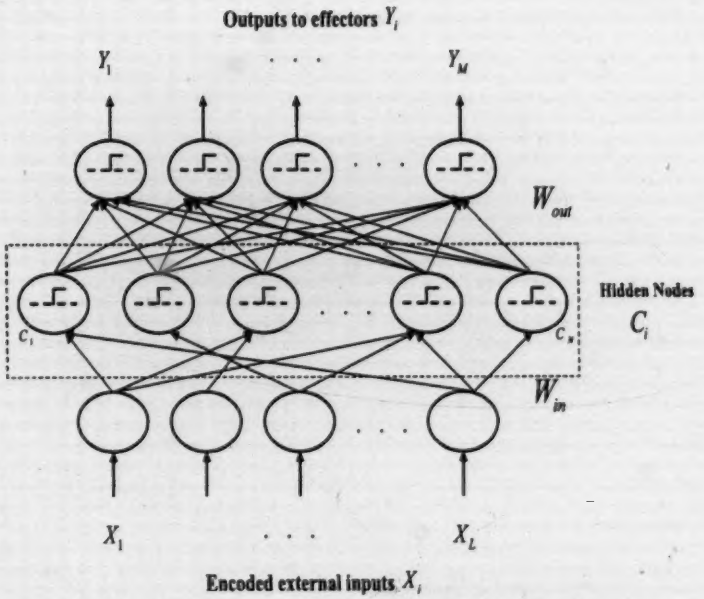

Fig. 7.1. A LCS as a Competitive NN

shows that mechanisms used in an LCS can be easily related to NN mechanisms. This study will discuss how the structure of an NN can benefit the LCS, and vice versa. The major problems include how to map the LCS into a feedforward NN, and to understand how the major subsystems of LCS (the performance, the reinforcement and the rule discovery) work in an NN. The biggest difference between an LCS and an NN appears to be the engagement of a GA. "*Although GAs have been used in training NNs, the approach used in LCSs is significantly different. Experimental results indicate that this LCS-inspired use of GAs creates a promising new learning technique for NNs.*" [98]. GAs in an LCS-like NN are discussed with respect to obstacle avoidance applications.

7.1.1 Mapping LCS into NN

Fig. 7.1 displays a simple MLP with three layers. This kind of architecture is part of a large feedforward NNs class with the neurons connected in a cascaded layers. All MLPs share

a common feature that all neurons in one layer are linked to all neurons in adjacent layers through unidirectional branches – the links broadcast and process information only in one direction, that is, the "forward direction" [34]. The input layer receives information from outside, a nonlinear processing is then performed in network and output pattern generates the control commands. To map the LCS into this NN, each classifier represents a hidden node (Fig. 7.1). The inputs to the NN are messages from the input interface: each bit of the input message is associated with an input node so that the number of input nodes corresponds to the length of classifier condition. Because of the application of an NN, it is possible to manipulate real numbers. This allows encoding the input message as an array of real numbers, each representing a current normalized sensor reading. The resulting network output values are associated with output nodes linked to various effectors.

Connection weights are bipolar numbers (either 1 or -1). Weights W_{in} between input and hidden nodes can be excitatory or inhibitory–that is, positive (+1) or negative (-1). W_{in} with +1 values correspond to 1 in the conditional part of the classifier. W_{in} with -1 values correspond to 0 in the classifier condition. The absence of a connection between an input node and a hidden node corresponds to a # symbol in the classifier condition.

A hidden layer is fully connected to the output layer with weights of +1 and -1 correspond to 1 and 0 in the action part of the classifier. The number of hidden layer nodes corresponds to the population size of the rule base N.

Since the classifier is described as a threshold node that outputs either 0 or +1, the activation function of each node in hidden or output layers is referred to as the hard limiter function $Y_h(U)$,

$$Y_h(U) = \begin{cases} 0, & \text{if } U<0 \\ \\ 1, & \text{otherwise} \end{cases} \tag{7.1}$$

The activation function can also be set as a binary sigmoid function, $Y_s(U)$,

$$Y_h(U) = \frac{1}{1+e^{-\alpha_s U}} \tag{7.2}$$

with large slope parameter α_s to ensure the threshold output of each classifier is either 0 or 1.

7.1.2 Execution of an LCS-Like NN

The Performance Subsystem:

The performance subsystem in an LCS-like NN is similar to the rule and message subsystem of an LCS. The operation cycle can be described as follows:

1. Detectors collect an environmental message and encode it as a real value string.
2. A population of rules is generated by a random process. Each classifier in the population is taken as an array that contains the weights of the NN. Similar to a regular classifier, it is divided into three parts. The first part is the conditional part corresponding to the weight values between the input layer and the hidden layer. The second part is the action part corresponding to weight values between hidden and output layers. The last part is a real value number representing the strength of that particular node.
3. A match set is formed from the classifiers in the population whose conditions match the input string.

   A Matching procedure is implemented in the hidden layer. The NN's detectors receive inputs from the environment and assemble the inputs into environmental messages. Inputs X_i are scaled by input layer neuron weights W_{in} , summed as U_{hid} and fed into each hidden neuron activation function, $Y_{hid}(U_{hid})$, a hard limiter activation function. The summations of weighted inputs are given by

$$U_{hid,n} = \sum_{i=1}^{I} X_i W_{in} \tag{7.3}$$

   where I is the number of inputs. In contrast to the matching process of traditional LCSs in which the condition and the message are matched bit by bit (Eq. 5.2), nodes with their thresholds exceeded are sent to a match set and then submitted to compete

in the auction procedure.

4. If no classifier matches the input, i.e. the match set is empty, a classifier is randomly generated through a so-called creation process.
5. The resulting network output values are translated into the output of the system through effectors.

The Reinforcement Subsystem:

An auction is performed among all the hidden layer nodes which have an antecedent that matches at least one of the environmental messages. With the matching pool determined, the auction commences. The basis of the competition is the strength of hidden layer nodes. Each node among the matching pool submits a bid. The node with the highest bid is selected as the winner of auction. The strength of the winning node at time t is then updated in a Michigan-style LCS mode (Eq. 4.13). The strength of all other nodes are updated in terms of Eq. 4.11 or Eq. 4.12.

In the LCS-like NN, it is assumed that only one active classifier is selected. A specific scaling method is then applied to hidden layer outputs Y_{hid} such that if selected as the winning node in the auction, the neuron $Y_{hid,win}$ outputs 1, while all other neurons output 0.

The outputs from the hidden layer neurons Y_{hid} are scaled by the output neuron weights W_{out}, summed as U_{out}

$$U_{out,m} = \sum_{n=1}^{N} Y_{hid,n} W_{out} \tag{7.4}$$

and fed into the output layer neurons with hard limiter functions, Y_{out}. N in Eq. 7.4 is the number of hidden nodes, corresponding to the population size of LCS.

The Rule Discovery Subsystem:

The back propogation (BP) algorithm is the learning process widely used to train NNs [34]. However, it is suitable only for supervised off-line learning scenarios. To provide online learning ability, a GA has been applied to search for weights either as a substitute for or as a supplement to the BP algorithm. These studies evolve a population of networks that are trained and updated in parallel. This approach is analogous to the Pittsburgh-type LCS and

may be impractical for large networks due to the high cost of network training.

In a conventional LCS, the GA evolves classifiers by combining existing classifiers with high-utility. Similarly, a GA can search for hidden layer nodes with new connection patterns in an LCS-like NN. Such a GA utilizes a population of hidden layer nodes in a single network rather than a population of networks, and thus significantly reduces the complexity of training. This is the main distinction between the use of GAs in an LCS-like NN and typical GA applications. In a typical application a GA determines a single individual that determines the solution. In an LCS-like NN a GA evolves a set of hidden nodes and therefore creates a set of individuals that jointly perform useful computation [98]. This coadaptive application of a GA varies greatly from common uses of GAs.

Since each hidden layer node represents a classifier associated with weights, the number of hidden nodes corresponds to the population size of a GA. All weights are defined and updated by a GA. In summary, a GA chooses higher-strength hidden layer nodes to reproduce and generates new individuals by their recombination and mutation.

7.2 An LCS-Like NN Controller

In using of an LCS-like NN for robot control system [60], the objective is to move the robot in its environment while avoiding obstacles.

During moving of robot, 16 sonar signals are continuously transformed into real number values in the range of [0,1] by a scaling process. That is, by getting the range of the last sonar reading for a given sonar, then dividing by the maximal range of this sonar (3000 *mm*). The decoding mechanism is the same as that introduced in section 5.1 to translate the outputs of an LCS-like NN into motor commands.

Since each hidden node represents a classifier rule, the length of the classifier is the total number of weights of each hidden node (22) plus an real strength value. The weights population is initialized using a uniform random distribution by filling all positions (conditional parts with -1, 1, and #; action parts with -1 and 1). A population size (number of hidden nodes) of 100 is used, and the initial strengths of all nodes are set to a common value

$S_i(0)$=100. An example of a complete hidden layer node configuration (corresponding to a classifier rule) is shown bellow

| 1 -1 # 1 -1 -1 -1 # 1 1 # -1 # # 1 -1 ‖ 1 -1 1 1 | -1 | 1 | 89 |

The credit assignment cycle starts with the matching process. The probability of selecting a winning node is equal to its bid divided by the sum of bids of all rules in the matching pool. To handle situations where no hidden node outputs unity, which means that no classifier rule matches the input, a "creation" process is used. The weights between the newly created node and all input nodes are all set to 1, except that # symbols are inserted with probability equal to the current percentage of all # symbols in the population. The action of the new node is set randomly, and its strength is set to the mean strength of the current population. To make room for the newly created node, one node is deleted from the population according to the probability distribution over the inverse of the strengths of all nodes. Next, the reinforcement component adjusts node strengths according to the payoff received from the environment after performing an action.

A GA is invoked after a number of iterations. To decrease the probability of disruption of good rules, the interval between GA executions grows as the robot learns. However, as mentioned in Chapter 6, merely applying this type of node-discovery process weakens the process of evolving correct actions. Assuming a collision occurs, the condition part of this action rule might be the only one to match the current environmental message: consequently, it will win every subsequent auction. The direct result is that the robot keeps colliding with obstacles until a GA introduces a new action. It may take long time to terminate the stalemate. To handle this problem, a special GA is proposed to apply exclusively to those rules that provide incorrect actions causing collisions. Only mutations with mutation rate $P_{m,spec}$ are applied to the action part, while the weights between the winning node and input nodes are all replaced by 1. Instead of being invoked after a number of iterations, this special GA is invoked right after each collision. This way, a new action with an updated conditional is created. If the action of the current winning node is appropriate, the robot

moves away from the obstacle. Otherwise, the robot will approach the obstacle again, and the special GA restarts. $P_{m,spec}$ is set to 0.3 which is much higher than typical to ensure creation of new actions.

The LCS-like NN controller has been applied to the same environments as the LCS controller. Fig. 7.2. is a flowchart of the LCS-like NN controller implementation.

Step 1: The connection weights of the NN are initialized by means of a random process. Each hidden node is linked to 16 input nodes and 6 output nodes. The weights between the hidden layer and the input layer are drawn from a ternary alphabet (-1, 0, +1) corresponding to (0, #, 1) in the condition of the LCS. The weights between the hidden layer and the output layer are drawn from the bipolar string (-1, +1) corresponding to (0, 1) in the action of the LCS. The initial strengths of all nodes are arranged to a common real value. In this step, a counter of collision, $Counter_{collision}$, and a counter of GA execution, n_{GA}, are defined and set to zero. T_i is pre-defined as the initial number of iterations after which a GA is triggered. Another variable, $Flag_{collision}$, is initialized as 0. $Flag_{collision}$ describes the robot status in the immediate past state: if collision occurred, $Flag_{collision}$=1, otherwise $Flag_{collision}$=0.

Step 2: Sonar readings are received and encoded as 16-bit real-valued strings by a scaling process.

Step 3: A matching process starts; only hidden nodes with their thresholds exceeded are sent to the match set, then submitted to compete in the auction procedure. If no rules in the population match the input, the creation process is engaged.

Step 4: A competition among the nodes in the match set is performed. A special scaling method is then applied to hidden nodes such that if being picked as the winner, the node outputs 1, while all other hidden nodes output 0. The winning node determines the network outputs which are decoded to motor commands, i.e., turning angle, velocity, and direction.

Step 5: The strengths of all the rules except the winning classifier are updated in terms of Eq. 4.14.

Step 6: The status of the robot is checked (e.g., for a stalled motor). If the left-motor or right-motor is stalled, the strength of the winning rule is updated by means of Eq. 4.13, in which R_i holds a fixed negative value. A special GA is applied to the winning classifier.

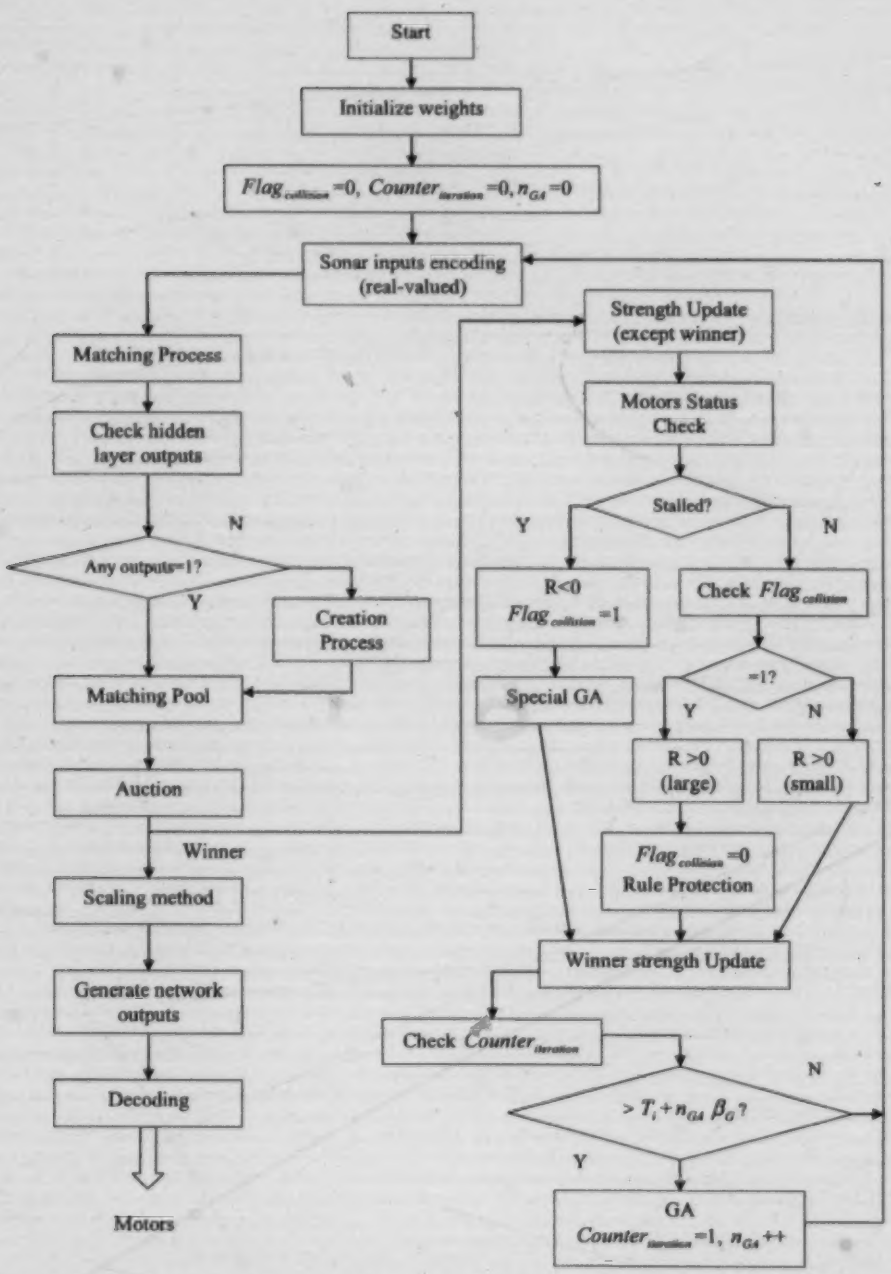

Fig. 7.2. Implementation cycle of obstacle avoidance by an LCS-like NN controller

$Flag_{collision}$ is set to 1 because a collision just occurred. If neither motor is stalled, the value of $Flag_{collision}$ is checked:

- R_i holds a fixed large positive value when $Flag_{collision} = 1$; the rule protection process starts and $Flag_{collision}$ is reset to 0 because no collision occurs in current iteration.
- R_i holds a fixed small positive value when $Flag_{collision} = 1$.

After the rewards value R_i is determined, the strength of the winning rule is updated by means of Eq. 4.13.

Step 7: The value of $Counter_{iteration}$ is checked. If $Counter_{iteration} \geq T_i + n_{GA}\beta_G$, where $\beta_G > 1$, the GA is triggered. $Counter_{iteration}$ is reset to 0, and n_{GA} is incremented by 1.

Step 8: The execution cycle proceeds from step 2.

Experiments show that the LCS-like NN endows the robot with learning ability. Fig. 7.3 and Fig. 7.4 show the robot's paths in pilaster and polygon environments, respectively, after the learning stage: no collisions occur.

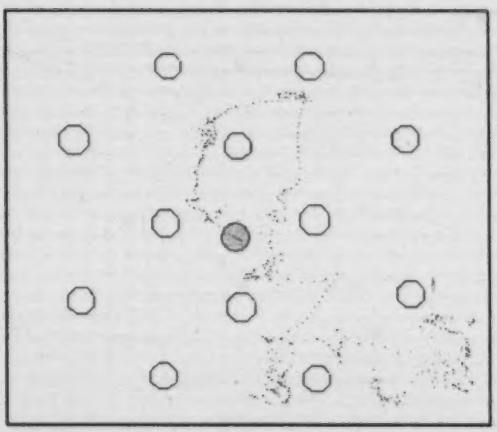

Fig. 7.3. Robot trajectory using LCS-like NN after the learning period (pilaster)

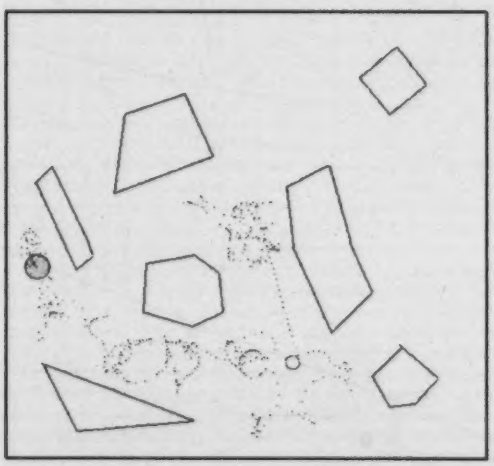

Fig. 7.4. Robot trajectory using LCS-like NN after the learning period (polygon)

Fig. 7.5 is a graph showing the cumulative number of iterations without collision during the learning phase in pilaster environment: the time between collisions is increasing, which indicates the robot is learning.

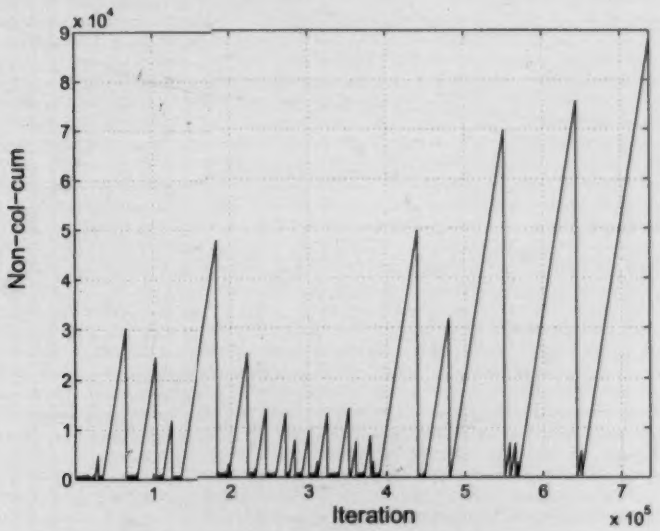

Fig. 7.5. Non-collision-cumulation (pilaster)

7.3 Parameters of the GA

The operation of an LCS-like NN is based on RL. The GA involved in its training provides the only possibility to introduce new actions and thus enrich the space searched by the RL mechanism. As such, performance of the GA is of paramount importance. Sensitivity of the GA performance to changes in its parameters is evaluated in the following sections.

7.3.1 The Number of Hidden Layer Nodes

The number of hidden layer nodes corresponds to the population size N of the GA, and it limits the number of classifiers available to the robot controller.

In general, as the number of environmental messages increases, the complexity of the learning task increases and there is a need for more classifiers to fulfil the task. However, the population size cannot be increased without limits to address this need due to associated demands for computing power and execution time. On the other hand, when the population size is too small, the system cannot distinguish between a sufficiently large number of classes and some useful rules (classifiers) can be replaced with new ones.

To study the effect of the GA population size on performance of the robot controller, a series of experiments was performed with population sizes from 100 to 500. Figures 7.6 to 7.9 show the cumulative number of iterations without collision for population sizes $N = \{100, 200, 300, 500\}$ in the polygon environment. The learning time is also summarized in Table 7.1.

Figures 7.5 and 7.6, show the results for pilaster and polygon environments, respectively, with the same population size ($N = 100$). It can be seen that the system takes longer to learn in the polygon environment. This result is expected because of the higher complexity of the polygon environment.

With N=200, the system stops learning around iteration $t = 3.6 \cdot 10^6$, earlier than in the case of N=100 ($t = 3.8 \cdot 10^6$): increasing N allows the system to accommodate more environmental messages and a fewer number of good rules are destroyed during learning. However, further increase in population size leads to an increase in NN execution time.

In other words, there is an optimal size of GA population. In general, this optimal size depends on the complexity of the environment in which learning occurs. In the pilaster environment used in this study, the optimal size was around 200.

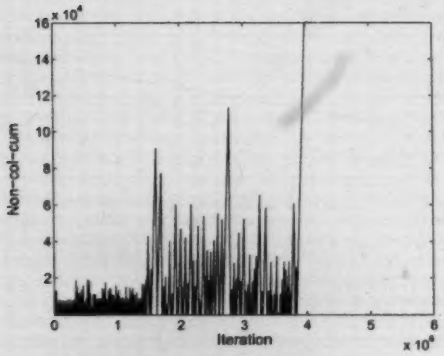

Fig. 7.6. Non-collision-cumulation with population size 100 (polygon).

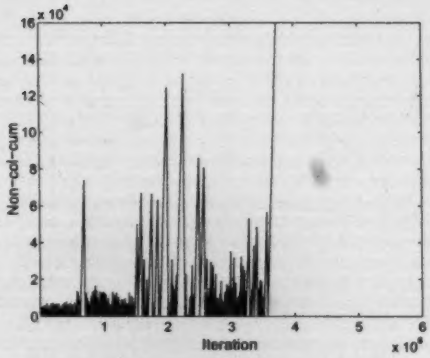

Fig. 7.7. Non-collision-cumulation with population size 200 (polygon).

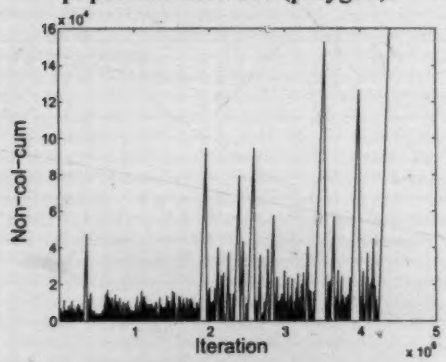

Fig. 7.8. Non-collision-cumulation with population size 300 (polygon).

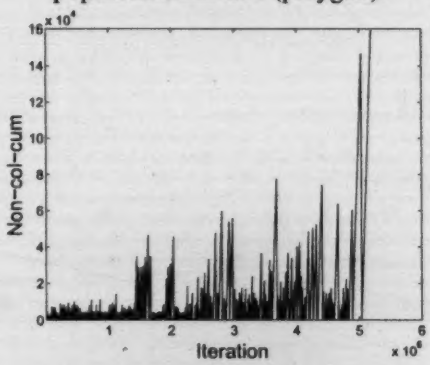

Fig. 7.9. Non-collision-cumulation with population size 500 (polygon).

7.3.2 Probability of Crossover

In a typical GA application, good on-line performance is associated with a high crossover rate P_c combined with a low mutation rate P_m. However, specific values of these probabilities depend on the problem to be solved [16].

A change of P_c affects the learning performance of the robot controller since the crossover

N	**Iterations**
100	$t = 3.8 \cdot 10^6$
200	$t = 3.6 \cdot 10^6$
300	$t = 4.3 \cdot 10^6$
500	$t = 5.1 \cdot 10^6$

TABLE 7.1

Learning time with various population size

operator is used to inherit parts of the genotype of well-performing parent rules. In the polygon environment, crossover probabilities of 0.4, 0.5, 0.6, 0.7, 0.8 and 0.9 have been tested.

Fig. 7.10 illustrates the effects of different values of the crossover rate P_c on learning time: learning time becomes shorter with increasing P_c. However, this change is less profound for $P_c > 0.7$. The reason is that such large crossover rates leave little space for keeping well-performing parents in the population, thus disturbing already largely converged classifiers.

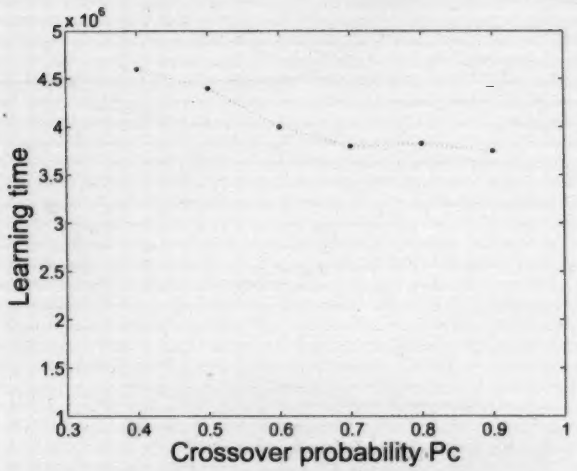

Fig. 7.10. The effect of different P_c on learning time

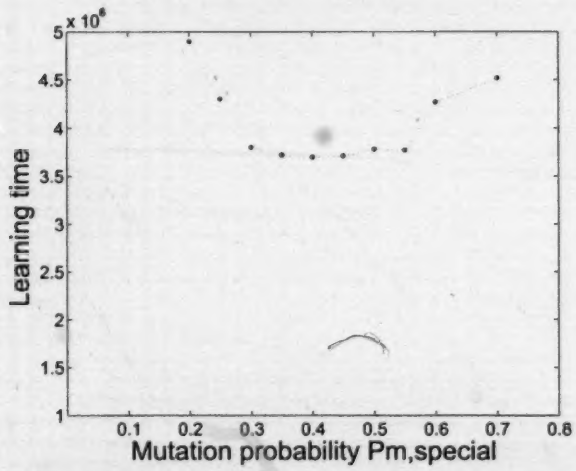

Fig. 7.11. The effect of different $P_{m,spec}$ on learning time

7.3.3 Probability of Mutation

In general, the mutation probability P_m impacts the frequency at which parts of the parent rules are mutated. Because a high mutation rate would disrupt well-performing rules in the population, the values of P_m are typically kept small. In this study, trials with values of P_m in range $[0, 0.1]$ demonstrate that sensitivity of the algorithm to the mutation rate is

negligible.

However, changes in $P_{m,spec}$ have a remarkable impact on the learning performance. Fig. 7.11 shows the effects of different values of $P_{m,spec}$ on learning time: learning time decreases with increasing $P_{m,spec}$ until it reaches a value of 0.4. When probability $P_{m,spec}$ is increased beyond 0.4, too much randomness is introduced to the population, which leads to instability of the classifier population and learning performance deteriorates.

Chapter 8

Conclusions and Future Work

In this chapter, the main conclusions of this thesis are presented, contents of the previous chapters are summarized, and recommendations for future work are briefly outlined.

8.1 Conclusions

This thesis describes obstacle avoidance subsystem of a mobile robot in an indoor environment utilizing sonar as the sensor of choice. The basic task of mobile robot navigation is to guide robot motion from a start location to an end location without collisions. Development of subsystem for obstacle avoidance is the main goal of this thesis. To achieve this goal, several novel learning models are proposed and, the learning ability and performance of these models are compared to existing systems.

LCS, a field of EC, is an active area of research. LCSs use a form of production rules (*if* condition *then* action), manipulated by GA and selected through an auction procedure. LCSs are flexible and LCS mechanisms of structural adaptation are superior to many other intelligent systems. Thus they are suitable for design of complex control systems. The novel LCS controller proposed in this thesis operates as follows: The system receives information about the environment, performs internal processing then provides motor commands to drive robot wandering. It then uses payoff from actions performed to modify the rule base; effectively, the system learns from experience.

Such an LCS possesses the ability to learn. Michigan-style LCSs can handle only simple environments – eventually the control system converges to a limit cycle exhibiting circular movement. To allow operation in more complex environments, extensions to Michigan-style LCS mechanisms are proposed by us to enhance performance.

- In order to allow the system to escape from local minima, GA is modified to respond to the number of execution cycles rather than the number of collisions. In addition, the interval of GA execution grows as the robot learns: at the beginning, after random initialization of the rule base, GA is invoked frequently to discover new well-performed rules. As the system learns more appropriate rules, application of GA becomes less frequent and the interval of its invocation is enlarged.
- Michigan-style LCS controllers often destroy well-performed rules that have been learned. To protect such useful rules, a large positive reward is granted to the rule whose action first leads a robot to move away from an obstacle after a collision. This rule is flagged and, as such, it will not be destroyed by GA in later generations.
- GA used in Michigan-style LCSs can yield new rules that are too general – leading to inability of the system to properly discriminate among specific environmental messages. To avoid this problem, a special genetic operator is applied only to the action part of the rule which resulted in a collision, while its condition part is replaced by current sonar inputs. This way, highly specialized rules are produced to replace unsuccessful rules that cause collisions.
- To speed up the learning process and maintain its stability, perceptual dynamics is introduced by augmenting the condition parts of classifiers. The response of the system at each time instant is then determined not only by the current state of the environment but also by past information in the form of delayed sonar inputs.

To further advance the flexibility and learning ability of LCS based obstacle avoidance, a fusion of LCS and NN is described in the last chapter of the thesis. By mapping an LCS onto an NN-based structure, the structural learning ability of an LCS can be combined with the continuous adaptive characteristics of an NN. An LCS-like NN controller was designed and implemented to decrease the learning time for obstacle avoidance behaviours.

In contrast to conventional NNs, the firing strength of each node in an LCS-like NN is used as a fitness value, receiving reward or penalty from the environment during the online learning process. Unlike conventional NN–GA studies that manipulate a population of networks, GA in an LCS-like NN operates on a single network that represents the entire population of classifiers. This approach introduces important computational advantages regarding learning time and speed of operation.

8.2 Future Work

Research on autonomous robots capable of self-learning is still far from industrial application and it is not yet possible to identify the most promising directions. A combination of different techniques has a greater chance to succeed in that individual techniques can reciprocally overcome their respective drawbacks resulting in hybrid systems that possess strong learning capabilities.

With regard to the topics treated in this thesis, future research is suggested in the following areas:

1. *LCS-like Recurrent NN Robot Control Systems*:

   Robots with reactive behaviours are unable to learn time-dependent tasks. This limitation can possibly be overcome by adding simple memory or by extending the sensor inputs loop along the time dimension. In this thesis, recurrent LCS is proposed by adding robot perceptual dynamics as inputs into a Michigan-style LCS. Adding a memory of past events to the LCS-like NN would be an interesting topic of future work. Various NN structures incorporate memory by using recurrent links that feed back state or output information. In some NN paradigms, outputs are fed back to the input layer as new inputs. Other NN paradigms feed back hidden unit states to context units. Such RNNs can be used in LCS-like robot control systems. The main advantage of mapping an LCS onto an RNN is that the resulting control system will have the ability to generate temporally continuous responses. Since the robot is constantly pushed around by its environment, reactive behaviours do not

allow the robot to take any initiative. In contrast, LCS-like RNN controllers would maintain past states. Consequently, responses to identical environmental stimuli can be distinct at different times, possibly preventing the robot from colliding with an obstacle or from getting stuck in a periodic orbit.

2. *Fuzzy LCS Robot Control Systems*:

   The fuzzy LCS robot controller combines the LCS ability of structural adaptation with the expressive power of FL.

   In LCS, the strength-based auction process in which rules compete for control is a relatively simple operation. In the fuzzy controller, however, all rules having non-zero activation values for a given input, play a part in the defuzzification that determines the final output. Therefore, the auction process must be modified. Accordingly, the reward or penalty received from the environment must be redistributed to all active rules that took part in determining the overall output.

   Two points regarding this work need to be considered:

   - LCS–based systems can be designed to operate either at the level of individual rules, (Michigan approach), or at the level of the entire rule-base (Pittsburgh approach). Computational requirements added using fuzzy rather than crisp or binary rules must be considered when selecting the approach suitable for design of fuzzy LCS-based systems.
   - LCSs are strength-based in the sense that a rule accrues strength while interacting with the environment. In XCS, accuracy of prediction is used to calculate classifier fitness. It has been shown that XCS can lead to evolution of optimally general rules. This approach could be adopted in fuzzy LCSs by substituting accuracy for strength to guide the evolution of fuzzy classifiers.

Bibliography

[1] Anderson, T. L and Donath, M, "Animal behaviour as a paradiam for developing robot autonomy," *Rob. Auton. Syst.*, vol. 6, pp. 145–168, 1990.

[2] Antonelo, E. A, "A neural reinforcement learning approach for behaviours acquisition in intelligent autonomous systems," Master's thesis, School of Information Science, Computer and Electrical Engineering, Halmstad University, 2006.

[3] Arkin, R, *Behaviour-Based Robotics*. The MIT Press, Cambridge, 1998.

[4] Arman, F and Aggarwal, J. K, "Model-based object recognition in dense-range images - a review," *ACM Computing Surveys*, vol. 25, pp. 5–43, 1993.

[5] Beer, R. D and Gallagher, J. C, "Evolving dynamic neural networks for adaptive behaviour," *Adaptive Behaviour*, vol. 1, pp. 91–122, 1992.

[6] Booker, L. B, Goldberg, D. E, and Holland, J. H, "Classifier systems and genetic algorithms," *Artificial Intelligence*, vol. 40, pp. 235–282, 1989.

[7] Braitenberg, V, *Vehicles: Experiments in Synthetic Psychology*. MIT Press, Cambridge, 1984.

[8] Braunstingl, R, Mujika, J, and Uribe, J. R, "A wall following robot with a fuzzy logic controller optimized by a genetic algorithm," in *Proc. IEE Int. Conf. Fuzzy Syst.*, pp. 77–82, Yokohama, Japan, March 1995.

[9] Brooks, R, "A robust layered control system for a mobile robot," *IEEE J. Robot. Autom.*, vol. 2, pp. 14–23, 1986.

[10] Carpenter, G. A and Grossberg, S, Eds., *Pattern Recognition by Self-Organizing Neural Networks.* MIT Press, Cambridge, 1991.

[11] Cazangi, R. R, Zuben, F. J. V, and Figueiredo, M, "A classifier system in real applications for robot navigation," in *The 2003 Congr. on Evol. Comput.*, pp. 574–580, Canberra, Australia, Dec. 2003.

[12] Chittaladakom, S, "Classifer system and genetic algorithm for rule-based global operation of canal gates," Ph.D. dissertation, Department of Biological and Irrigation Engineer, Utah State University, 1996.

[13] Cliff, D, Husbands, P, and Harvey, I, "General visual robot controller networks via artificial evolution," in *Proc. of the Society of Photooptical Instrum. Engineers Conf.*, pp. 1–11, Boston, USA, Aug. 1993.

[14] Cox, L. J and Wilfong, G. T, *Autonomous Robot Vehicles.* Spinger-Verlag, New York, 1990.

[15] Darwin, C, *The Origin of Species.* John Murray, Albemarle Street, London, 1859.

[16] Davis, L, *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.

[17] DeJong, K. A, "Using genetic algorithms to learn task programs: the Pitt approach," *Machine Learning*, vol. 3, pp. 2–3, 1988.

[18] Dorigo, M and Colombetti, M, Eds., *Robot Shaping: an Experiment in Behaviour Engineering.* MIT Press, Cambridge, 1998.

[19] Dubrawski, A and Crowley, J. L, "Learning locomotion reflexes: a self-supervised neural system for a mobile robot," *Rob. Auton. Syst.*, vol. 12, pp. 133–142, 1994.

[20] Dubrawski, A and Crowley, J. L, "Self-supervised neural system for reactive navigation," in *Proc. IEEE Int. Conf. Rob. and Autom.*, pp. 2076–2082, San Diego, USA, May 1994.

[21] Dumitrescu, D, Lazzerino, B, Jain, L. C, and Dumitrescu, A, *Evolutionary Computation*. CRC Press, Boca Raton, 2000.

[22] Eiben, A. E and Smith, J. E, Eds., *Introduction to Evolutionary Computing*. Springer, New York, 2003.

[23] Everett, H. R, *Sensors for Mobile Robots: Theory and Application*. A. K. Peters, Natick, 1995.

[24] Fayad, C and Webb, P, "Optimized fuzzy logic based algorithm for a mobile robot collision avoidance in an unknown environment," in *7th European Congress on Intelligent Techniques and Soft Computing*, pp. 1–6, Aachen, Germany, 1999.

[25] Filliat, D, Kodjabachian, J, and Meyer, J. A, "Incremental evolution of neural controllers for navigation in a 6 1egged robot," in *Proc. 4th Int. Symp. on Artif. Life and Rob.*, Oita, Japan, 1999.

[26] Floreano, D, Godjevac, J, Martinoli, A, Mondada, F, and Nicoud, J. D, "Design, control, and applications of autonomous mobile robots," Micro-computing Laboratory Swiss Federal Institute of Technology in Lausanne LAMI-INF-EPFL, CH-1015 Ecublens., Tech. Rep., 1999.

[27] Fogel, L. J, Owens, A. J, and Walsh, M. J, *Artificial Intelligence through Simulated Evolution*. John Wiley and Sons, New York, 1966.

[28] Foster, K. Y, "An investigation of the use of past experience in single and multiple agent learning classifier systems," Ph.D. dissertation, Faculty of Information and Communication Technologies, Swinburne University of Technology, 2005.

[29] Gandikota, V, "Effect of missing information on a classifier system," Master's thesis, Department of Computer Science, University of Nevada, 1998.

[30] Goldberg, D, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Boston, 1989.

[31] Grossberg, S, Ed., *Studies of Mind and Brain: Neural Principles of Learning, Perception, Development, Cognition and Motor Control*. Reidel, Boston, 1982.

[32] Grossberg, S, Ed., *Neural Networks and Natural Intelligence*. MIT Press, Cambridge, 1989.

[33] Gruau, F, "Automatic definition of modular neural networks," *Adaptive Behaviour*, vol. 2, pp. 151–183, 1995.

[34] Ham, F. M and Kostanic, I, *Principles of Neurocomputing for Science and Engineering*. McGraw-Hill Inc., New York, 2000.

[35] Harvey, I, "Species adaptation genetic algorithm: the basis for a continuing SAGA," in *Proc. 1st European Conf. on Artif. Life: Toward a Practice of Auton. Syst.*, MIT Press/Bradford Books, Cambridge, pp. 346–354, 1992.

[36] Harvey, I, "Evolving robot consciousness: The easy problems and the rest," in *Evolving Consciousness*. John Benjamins, pp. 205–219, 2002.

[37] Harvey, I, "Artificial evolution: A continuing SAGA," in *Proc. Int. Symp. Evol. Rob.: From Intelligent Robotics to Artificial Life*, pp. 94–112, 2001.

[38] Harvey, I, Husbands, P, Cliff, D, Thompson, A, and Jakobi, N, "Evolutionary robotics: the Sussex approach," *Rob. Auton. Syst.*, vol. 20, pp. 205–224, 1996.

[39] Harvey, I, Husbands, P, Cliff, D, Thompson, A, and Jakobi, N, "Evolutionary robotics at Sussex," in *Robotics and Manufacturing: Recent Trends in Research and Applications*, vol. 6, pp. 293–298, 1996.

[40] Holland, J. H, *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, 1975.

[41] Holland, J. H, "Adaptive algorithms for discovering and using general patterns in growing knowledge bases," *International Journal for Policy Analysis and Information Systems*, vol. 4, pp. 245–268, 1980.

[42] Holland, J. H, "Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems," in *Machine Learning: An Artificial Intelligence Approach (Vol 2)*. Tioga Publishing Company, Palo Alto, pp. 593–623, 1986.

[43] Holland, J. H and Reitman, J. S, *Cognitive Systems Based on Adaptive Algorithms*. Pattern-Directed Inference Systems, Academic Press, New York, 1978.

[44] Hornik, K, Stinchcombe, M, and White, H, "Multiplayer feedforward networks are universal approximators," *Neural Networks*, vol. 2(5), pp. 359–366, 1989.

[45] Hurst, J, Bull, L, and Melhuish, C, "Zcs and tcs learning classifier system controllers on real robots," University of the West of England, UK, Tech. Rep., 2002.

[46] Husbands, P, Harvey, I, Cliff, D, Thompson, A, and Jakobi, N, "Artificial evolution of control systems," in *Proc. 2nd Int. Conf. on Adaptive Computing in Engineering Design and Control '96 I.C. Parmee (ed)*, pp. 41–49, Plymouth, UK, 1996.

[47] Husbands, P, Harvey, I, Jakobi, N, Thompson, A, and Cliff, D, "Evolutionary robotics," in *Handbook of Evolutionary Computation*, Back, T, Fogel, D, and Michalewicz, Z, Eds. Oxford University Press/Institute of Physics, Chap. G3.7, 1997.

[48] Husbands, P and Meyer, J.-A, "Evolutionary robotics: a survey of applications and problems in evolutionary robotics," in *First European Workshop, EvoRbot98*, ser. Lecture Notes in Computer Science, vol. 1468, pp. 1–21, London, UK, 1998.

[49] Jakobi, N, "Running across the reality gap: Octopod locomotion evolved in a minimal simulation," in *First European Workshop, EvoRbot98*, ser. Lecture Notes in Computer Science, vol. 1468, pp. 39–59, 1998.

[50] Jakobi, N, Husbands, P, and Harvey, I, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *Advances in Artificial Life: Proc. 3rd Euro. Conf. Artif. Life*, ser. Lecture Notes in Computer Science, Springer-Verlag, vol. 929, pp. 704–720, 1995.

[51] Kassahun, Y, "Towards a unified approach to learning and adaptation," Ph.D. dissertation, University of Bremen, 2006.

[52] Kosko, B, Ed., *Neural Networks and Fuzzy Systems*. Prentice-Hall, Santa Cruz, 1992.

[53] Kovacs, T, "Evolving optimal populations with XCS classifier systems," School of Computer Science, University of Birmingham, UK, Tech. Rep., CSRP-96-17, 1996.

[54] Kovacs, T, "XCS classifier system reliably evolves accurate, complete, and minimal representations for boolean functions," in *Soft Computing in Engineering Design and Manufacturing*, Roy, Chawdhry, and Pant, Eds. Springer-Verlag, pp. 59–68, 1997.

[55] Koza, J. R, "Evolution of subsumption using genetic programming," in *Proc. 1st Euro. Conf. Artif. Life: Toward a Practice of Aut. Syst.*, MIT Press, Cambridge, pp. 110–119, 1992.

[56] Lanzi, P. L, Stolzmann, W, and Wilson, S. W, Eds., *Learning Classifier Systems: From Fundations to Applications*. Vol. 1813 of LNAI, Springer-Verlag, Berlin, 2000.

[57] Lanzi, P. L and Wilson, S. W, "Optimal classifier system performance in non-markovian environments," Illinois Genetic Algorithms Laboratory University of Illinois at Urbana-Champaign, Tech. Rep., 1999.

[58] Lewis, A, Fagg, A. H, and Bekey, G. A, "Genetic algorithms for gait synthesis in a hexapod robot," *Recent trends in mobile robots*, pp. 317–331, 1993.

[59] Li, S, Musilek, P, and Wyard-Scott, L, "Enhanced learning classifier system with perceptual memory for robot navigation," *Research Report, FACIA LAB*, University of Alberta, 2005.

[60] Li, S, Musilek, P, and Wyard-Scott, L, "Learning classifier system like neural network for robot navigation," *Research Report, FACIA LAB*, University of Alberta, 2006.

[61] Lukins, T. C, "Dynamically developing novel and useful behaviours a first step in animat creativity," Master's thesis, School of Artificial Intelligence School of Informatics, University of Edinburgh, 2002.

[62] Martin, M. C, "The simulated evolution of robot perception," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, 2001.

[63] McAulay, A. D and Oh, J. C, "Improved leaning in genetic rule-based classifier system," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, no. 1, pp. 152–159, 1994.

[64] Meng, Q. C, Feng, T. J, Chen, Z, Zhou, C. J, and Bo, J. H, "Genetic algorithms encoding study and a sufficient convergence condition of gas," in *Proc. IEEE SMC '99 Int. Conf.*, pp. 649–652, Tokyo, Japan, Oct. 1999.

[65] Michalewicz, Z, *Genetic algorithms + data structures = evolution programs*. Springer-Verlag, Berlin, 1994.

[66] Michalski, R. S, Carbonell, J. G, and Mitchell, T. M, *Machine Learning: An Artificial Intelligence Approach*. Tioga Publishing Company, Palo Alto, 1983.

[67] Michalski, R. S, Carbonell, J. G, and Mitchell, T. M, Eds., *Machine Learning: An Artificial Intelligence Approach (Vol 2)*. Tioga publishing company, Palo Alto, 1986.

[68] Mitchell, T. M, Ed., *Machine Learning*. McGraw-Hill, Boston, 1997.

[69] Moore, W and Harvey, I, "Evolution of a non-linear robot controller in robotics and applications," in *Proc. IASTED Int. Conf. on Rob. and Appl.*, Santa Barbara, USA, Oct. 1999.

[70] Moussi, L. N, Gudwin, R. R, Zuben, F. J. V, and Madrid, M. K, "Neural networks in classifier systems (NNCS): An application to autonomous navigation," in *Advances in Signal Processing, Robotics and Communications, Electrical and Computer Engineering Series, WSES Press*, pp. 256–262, Danvers, 2001.

[71] Moussi, L. N, Gudwin, R. R, Zuben, F. J. V, and Madrid, M. K, "A simulator using classifier systems with neural networks for autonomous robot navigation," in *IJCNN'02 - 2002 Int. Joint Conf. on Neural Networks- World Conf. on Comput. Intell.*, pp. 501–506, Honolulu, USA, May 2002.

[72] Musilek, P, "Principles of autonomous mobile robot control," *Neural Networks World*, vol. 3, pp. 249–260, 1993.

[73] Musilek, P, "Artificial neural networks in robot navigation," *Neural Network World*, vol. 5, pp. 929–994, 1995.

[74] Musilek, P, Li, S, and Wyard-Scott, L, "Enhanced learning classifier system for robot navigation," in *IEEE/RSJ Int. Conf. on intell. Rob. Syst.*, pp. 3390–3395, Edmonton, Canada, Aug. 2005.

[75] Narendra, K and Thathachar, M. A. L, *Learning automata, an introduction*. Prentice Hall International, Englewood Cliffs, 1989.

[76] Nelson, A. L, Grant, E, and Lee, G, "Using genetic algorithms to capture behavioural traits exhibited by knowledge based robot agents," in *Proc. ISCA 15th Int. Conf.: Computer Applications in Industry and Engineering (CAINE-2002)*, pp. 92–97, , San Diego, USA, Nov. 2002.

[77] Ng, K. C and Trivedi, M. M, "A neuro-fuzzy controller for mobile robot navigation and multirobot convoying," *IEEE Trans. Syst., Man, Cybern. B*, vol. 28, pp. 829–840, 1998.

[78] Nillson, N. J, Ed., *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill, New York, 1971.

[79] Nolfi, S and Floreano, D, Eds., *Evolutionary Robotics*. The MIT Press, Cambridge, 2000.

[80] Nolfi, S, Floreano, D, Miglino, O, and Mondada, F, "How to evolve autonomous robots: Different approaches in evolutionary robotics," in *Proc. IV Int. Workshop on Artificial Life*, pp. 190–197, MIT Press/Bradford Books, Cambridge, 1994.

[81] Pinto, J, "The robots are coming," *Automation World*, p. 62, 2006.

[82] Pipe, A. G and Carse, B, "First results from experiments in fuzzy classifier system architectures for mobile robotics," in *Proc. 7th Int. Conf. Parallel Problem Solving from Nature*, pp. 578–587, London, UK, 2002.

[83] Pipe, A. G, Carse, B, and Malliris, E, "Experiments on a Pittsburgh-style fuzzy classifier system for mobile robotics," in *Proc. 2002 IEEE Int. Symp. on Intell. Contr.*, pp. 61–62, Vancouver, Canada, Oct. 2002.

[84] Pomerleau, D. A, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in Neural Information Processing Systems*, Morgan Kaufmann, San Fransisco, pp. 305–313, 1989.

[85] Pomerleau, D. A, "Neural network based autonomous navigation," in *Vision and Navigation: The CMU Navlab*, Thorpe, C, Ed. Kluwer Academic, Norwell, pp. 83–92, 1990.

[86] Pomerleau, D. A, "Efficient training of an artificial neural networks for autonomous navigation," *Neural Comput.*, vol. 3, no. 1, pp. 88–97,1991.

[87] Pomerleau, D. A, Gusciora, G. L, Touretzky, D. S, and Kung, H. T, "Neural network simulation at warp speed: How we got 17 million connections per second," in *Proc. IEEE Int. Join. Conf. on Neural Networks*, pp. 143–150, July 1988.

[88] Port, R. F, "The dynamical systems hypothesis in cognitive science," in *Encyclopedia of Cognitive Science*, Lockyer, A, Ed. MacMillan, 2002.

[89] Rechenberg, I, *Evolutionsstrategie*. Friedrich Frommann Verlag, Stuttgart, 1973.

[90] Reignier, P, "Fuzzy logic techniques for mobile robot obstacle avoidance," *Rob. Auton. Syst.*, vol. 12, no. 3, pp. 143–153, 1994.

[91] Richards, R. A, "Zeroth-order shape optimization utilizing a learning classifier system," Ph.D. dissertation, Mechanical Eng. Dept., Stanford University, 1995.

[92] Rylatt, M, Czarnecki, C, and Routen, T, "Connectionist learning in behaviour-based mobile robots: A survey," *Artificial Intelligence Review*, vol. 12, pp. 445–468, 1998.

[93] Saffiotti, A, Ruspini, E. H, and Konolige, K, "Blending reactivity and goal-directedness in a fuzzy controller," in *Proc. 2th IEEE Int. Conf. on Fuzzy Syst.*, pp. 134–139, San Francisco, USA, 1993.

[94] Schmidt, A and Bandar, Z, "A modular neural network architecture with additional generalisation abilities for large input vectors," in *Proc. 3rd Int. Conf. Artif. Neural Networks and Genetic Algorithms (ICANGA97)*, pp. 40–43, Norwich, UK, April 1997.

[95] Silvayz, C, Crisostomox, M, and Ribeiroy, B, "A neural modular learning architecture for navigating the nomad mobile robot without knowledge of the environment," in *8th Int. Conf. Inform. Proces. and Managem. of Uncertainty in Knowledge Based Syst.*, pp. 1199–1204, Madrid, Spain, July 2000.

[96] Simoes, E, "Developement of an embedded evolutionary controller to enable conllision-free navigation of a population of autonomous mobile robots," Ph.D. dissertation, The University of Kent at Canterbury, 2000.

[97] Simon, H. A, Ed., *The Sciences of the Artificial.* MIT Press, Cambridge, 1996.

[98] Smith, R. E and Cribbs, H. B, "Is a learning classifier system a type of neural network?" *Evolutionary Computation 2*, vol. 1, pp. 19–36, 1994.

[99] Smith, T. M, Husbands, P, and O'Shea, M, "Evolvability and analysis of robot control networks," in *Proc. 7th Int. Conf. Simulation of Adaptive Behaviour*, pp. 349–350, Edinburgh, UK, Aug. 2002.

[100] Sommer, G, Hailu, G, and Bruske, J, "Navigation project," Tech. Rep., accessed on February 1, 2007.

[101] Stolzmann, W, "Latent learning in khepera robots with anticipatory classifier systems," in *Proc. 99 Genetic and Evolutionary Comput. Conf.*, pp. 290–297, Orlando, USA, 1999.

[102] Studley, M, "Learning classifier systems for multi-objective robot control," Ph.D. dissertation, Faculty of Computing, Engineering and Mathematics, University of the West of England Coldharbour Lane, Frenchay, Bristol BS16 1QY, UK., 2005.

[103] Sugeno, M and Nishida, M, "Fuzzy control of a model car," *Fuzzy Sets and Systems*, vol. 16, pp. 103–113, 1985.

[104] Thongchai, S and Kawamura, K, "Application of fuzzy control to a sonar-based obstacle avoidance mobile robot," in *Proc. 2000 IEEE Int. Conf. Control Applications*, pp. 425–430, Anchorage, USA, 2000.

[105] Thongchai, S, Suksakulchai, S, Wilkes, D. M, and Sarkar, N, "Sonar behaviour-based fuzzy control for a mobile robot," in *Proc. IEEE Int. Conf. Syst., Man, and Cybern.*, pp. 3532–3537, Nashville, USA, Oct. 2000.

[106] Tijsseling, A. G, "Sequential information processing using time-delay connections in ontogenic calm networks," *IEEE Trans. Neural Netw.*, vol. 16, pp. 145–159, 2005.

[107] Tuci, E, Harvey, I, and Quinn, M, "Evolving integrated controllers for autonomous learning robots using dynamic neural networks," in *Proc. 7th Int. Conf. on the Simulation of Adaptive Behaviour (SAB'02)*, pp. 282–291, Cambridge, USA, August 2002.

[108] Tuci, E, Quinn, M, and Harvey, I, "Evolving fixed-weight networks for learning robots," in *Proc. Congress on Evolutionary Computation (CEC)*, vol. 2, pp. 1970-1975, Honolulu, USA, 2002.

[109] Vasilyev, A, "Synergetic approach in adaptive systems," Master's thesis, Transport and Telecommunication Institute, Riga, Latvia, 2002.

[110] Walter, W, *The Living Brain*. W. H. Norton, New York, 1953.

[111] Whitbrook, A, "An idiotypic immune network for mobile robot control," Master's thesis, School of Computer Science and Information Technology, University of Nottingham, 2005.

[112] Wilson, S, "ZCS: A zeroth level classifier system," *Evolutionary Computation*, vol. 2, pp. 1–18, 1994.

[113] Wilson, S, "Classifier fitness based on accuracy," *Evolutionary Computation*, vol. 3, pp. 149–175, 1995.

[114] Wilson, S. W, "Classifier systems and the animat problem," *Machine Learning*, vol. 2, pp. 199–228, 1987.

[115] Xie, M, Ed., *Fundamentals of Robotics: Linking Perception to Action*. World Scientific, Singapore, 2003.

[116] Yen, J and Pfluger, N, "A fuzzy logic based extension to Payton and Rosenblatt's command fusion method for mobile robot navigation," *IEEE Trans. Syst., Man, Cybern.*, vol. 25, pp. 971–978, 1995.

[117] Yen, J and Pflunger, N, "A fuzzy logic based robot navigation system," in *AAAI Fall Symposium on Mobile Robot Navigation*, pp. 195–199, Cambridge, USA, Oct. 1992.

[118] Zadeh, L. A, "Fuzzy sets," *Information and Control*, vol. 8, pp. 338–353, 1965.

[119] Zavlangas, P. G, Tzafestas, S. G, and Althoefer, K, "Fuzzy obstacle avoidance and navigation for omnidirectional mobile robots," in *Proc. European Symposium on Intelligent Techniques (ESIT 2000)*, pp. 375–382, Aachen, Germany, Sept. 2000.